REPORT DOCUMENTATION PAGE

Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 20, 1998	Final Report (Phase II Option)

4. TITLE AND SUBTITLE
Demonstration of Iconic, Graphic, Data Flow Programming for High Performance Real Time Workstations - Phase II Option Final Report

5. FUNDING NUMBERS
C N00014-96-C-0208
Contract Line Item No. 0004
Data Item No. B002

6. AUTHOR(S)
N. Carl Ecklund, Technical Director, MCCI

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Management Communications and Control, Inc. (MCCI)
2000 North 14th Street
Suite 220
Arlington, VA 22201

8. PERFORMING ORGANIZATION REPORT NUMBER
MCCI-98-ONR-002

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Richard Lau, Prog. Off.
Office of Naval Research
Code 311
BCT #1, 800 N. Quincy St.
Arlington, VA 22217-5660

Mr. R. Cockerill, Monitor
SPAWAR PEO (Submarine Prog.)
Naval Sea Systems Command
2531 Jefferson Davis Hwy.
Arlington, VA 22242-6168

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

19980603 035

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

As the Navy adopts COTS processors for signal processing systems, there is a need for desktop workstations with either embedded or attached COTS processing systems to act as platforms for development and maintenance of algorithms for embedded signal processing systems. Under the basic Phase II program, MCCI has integrated the PGM based Autocoding Toolset developed under the DARPA funded RASSP program with rapid prototyping tools developed under this SBIR. In the Phase II option program, MCCI demonstrated use of the PGM programming environment for high performance workstations by implementing five (5) application graphs representative of submarine sonar passive broadband and narrowband processing. Requirements specifications were implemented as PGM graphs and autocoded into executable code. Test signal generators and displays were developed to allow exercising the applications on the company's high performance multi-computer workstation. The report describes each application, PGM graphs, and examples of output from the autocoded implementations.

14. SUBJECT TERMS Signal Processing Workstation, Data Flow Graphs, DSP, Parallel Processing, Processing Graph Method (PGM), Autocoding, Sonar Signal Processing, Graphical Programming, Submarine Sonar, Iconic

15. NUMBER OF PAGES 57

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Report Number: MCCI-98-ONR-002

Demonstration of Iconic, Graphic, Data Flow Programming for High Performance Real Time Workstations Phase II Option Final Report

May 20, 1998

Prepared For:
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5660

Prepared by:
Management Communications and Control, Inc. (MCCI)
2000 North Fourteenth Street, Suite 220
Arlington, VA 22201-2555
(703) 522 - 7177

Under Contract Number:
N00014-96-C-0208

Demonstration of Iconic, Graphic, Data Flow Programming for High Performance Real Time Workstations Phase II Option Final Report

Table of Contents

Demonstration of Iconic, Graphic, Data Flow Programming for High Performance Real Time Workstations Phase II Option Final Report

List of Figures

List of Symbols, Abbreviations, Acronyms

AG - Application Generator
API - Application Programmer's Interface
CP GUI - Command Program Graphical User Interface
CPI - Command Procedure Interface
EAG - Equivalent Application Graph
GIP - Graph Instantiation Parameter
GrTT - Graph Translation Tool
GSMP - Graph Execution Simulation on Multiple Processors
GUI - Graphical User Interface
GV - Graph Variable
MPID - Multi Processor Interface Description
MPIDGen - Multi Processor Interface Description Generator
PB - Partition Builder
PGM - Processing Graph Method
SPGN - Signal Processing Graph Notation
SRS - Software Requirements Specification
SRTS - Static Run-Time System

Demonstration of Iconic, Graphic, Data Flow Programming for High Performance Real Time Workstations Phase II Option Final Report

Introduction

MCCI developed several benchmark applications in the Phase II option task to demonstrate the PGM workstation programming environment. Benchmarks were selected from the requirements of potential U.S. Navy users. The benchmarks included signal processing and command and control requirements. Application signal processing requirements were specified as PGM graphs using DSPGraph, a graph entry/editor tool. The PGM graphs were then translated using the MCCI Autocoding Toolset. The target chosen was a Mercury Computer platform consisting of a board with four I860s and a board with four PowerPCs. The primary product of the Autocoding Toolset translation is source code for the application.

The translation consisted of the following steps:

1. An Equivalent Application Graph (EAG) and a set of partition graphs were generated for each signal processing application PGM graph using the Partition Builder component of the MCCI Autocoding Toolset.

2. Partition graphs were translated to MPIDs using the MPIDGen component of the MCCI Autocoding Toolset.

3. Individual MPIDs were tested using the MPID Test Environment. This was performed on both the target platform and on a Sun workstation.

4. The applications were translated using the Application Generator component of the MCCI Autocoding Toolset. This step generates a node task wrapper for each node in the EAG, a thread manager for each processor on which at least one node of the EAG will execute, and a description of the application for use by the Graph Manager component of the Static Run-Time System. The Static Run-Time System is a set of services provided for data flow graph execution of the EAG and for external control of the application.

5. The applications were tested. During this phase of development, external control was via the command Program Graphical User Interface (CP GUI) component of the MCCI Autocoding Toolset. Command macros (sequences of Command Program application interaction commands) required to meet control requirements were generated using the CP GUI interactively with the application. Control of the application from the CP GUI was demonstrated.

6. The execution performance of one application graph was evaluated with the GSMP performance simulation component of the MCCI Autocoding Toolset. During this evaluation, the inefficient execution of one Domain Primitive was observed. The Domain Primitive implementation was subsequently modified to improve execution efficiency. This highlighted the usefulness of performance simulation.

The application graphs that were developed are discussed in the next section. The graphs were developed by a user familiar with signal processing. The MCCI Autocoding Toolset translations were performed by a new user of the toolset. As part of the process of learning to use the MCCI Autocoding Toolset by the new user, user manuals were reviewed for completeness and accuracy. The user manuals were revised based on comments by the new user.

Demonstration Graphs

As part of the Phase II Option program, MCCI implemented some signal processing graphs. These graphs are:

Broadband array
Broadband array pair
Broadband array cross correlation
Broadband array cross correlation pair
Narrowband baseline
Narrowband high frequency
Narrowband medium frequency.

In implementing each of the graphs, MCCI started from a description of the processing in a Software Requirements Specification (SRS). This SRS contained block diagrams and both textual and mathematical descriptions of the processing. With this level of documentation, it was rather easy with only a few exceptions to develop the data flow graphs using signal processing routines from the Domain Primitive library. The exceptions arose from ambiguous textual descriptions that did not include mathematical descriptions.

Some of the processing was common to more than one graph, and some processing was duplicated (except for parameter values) within graphs. This duplication of processing is readily expressed in PGM as subgraphs. The use of subgraphs in this manner can substantially reduce the development and unit testing time.

Details of the individual graphs are provided in subsequent sections.

Development Effort

A summary of the development effort in terms of hours is contained in Table 1. As shown in the table, a large part of the development effort was associated with developing signal generation simulators and with developing displays.

Category	Hours
Graph Development	223
Autocoding	93
Graph Testing	237
MPID Testing	168
I/O Procedure Development	184
Display Development	193
CP Development	37
Primitive Maintenance	61

Table 1. Development Effort Hours

The graphs were developed using DSPGraph, a tool developed for the U.S. Navy by Lucent Technologies by a person experienced in signal processing. The Navy has made this tool available, and MCCI has permission to include this tool as "freeware" in deliveries of the MCCI Autocoding Toolset. The output from DSPGraph is Signal Processing Graph Notation (SPGN) which is the language for graphs developed under the Processing Graph Method (PGM).

A new user of the MCCI Autocoding Toolset performed the translations. As part of this effort, the user manuals were reviewed for completeness and accuracy. The user manuals were revised based on comments by the new user. The hours required for the review and revision of the manuals are not included in Table 1.

In testing the graphs, the CP GUI tool developed under Phase II was used to control the application graphs, including starting and stopping the graphs, and initializing, starting, and stopping the I/O Procedures. By generating macros for common operations such as application initialization, which includes creating the queues required to connect the I/O Procedures to the graph being tested, and initializing the I/O Procedures, starting the Output Procedure and starting the graph, considerable time savings were realized, demonstrating the usefulness of the CP GUI tool.

Broadband Array

The top level graph for the processing associated with the Broadband Array is shown in Figure 1.

Overview of the Processing

The Broadband Array processing is intended to detect broadband signals. A slice of the spectrum is examined for energy above a certain level. As shown in Figure 1, the Broadband Array processing consists of filtering, short term averaging, noise estimation and normalization, and integration within a beam. The input data has been beamformed into `NBEAMS` beams (where `NBEAMS` is equal to 55) and is time domain data. It has been assumed that `NPTS` samples of

beamformed time data from each beam have been concatenated into one data stream, where NPTS is equal to 128. Thus, the input data can be thought of as a matrix of 55 rows by 128 columns.

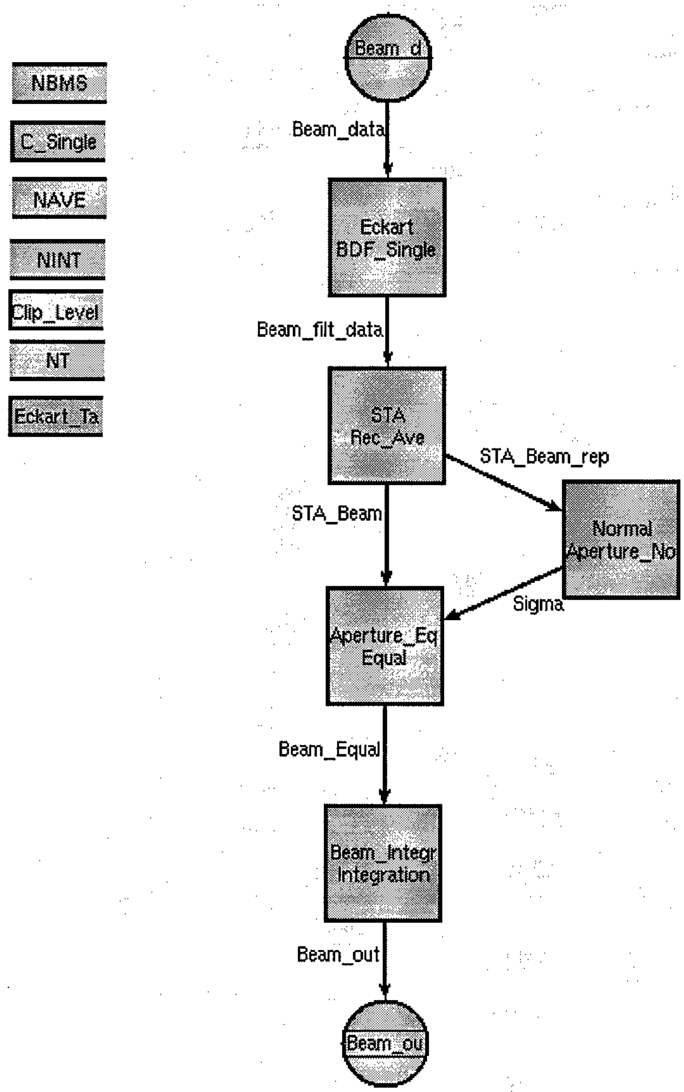

Figure 1. Broadband Array Processing

The SPGN for the Broadband Array processing is shown below.

```
%GRAPH( LRS
    VAR      = Clip_Level : FLOAT
    INPUTQ   = Beam_data : FLOAT
    OUTPUTQ = Beam_out : FLOAT )
 %GIP( NBMS : INT INITIALIZE TO 55 )
 %GIP( NAVE : INT INITIALIZE TO 128 )
 %GIP( NINT : INT INITIALIZE TO 8 )
 %GIP( NT : INT INITIALIZE TO 3 )
 %GIP( C_Single_1 : FLOAT ARRAY (3) INITIALIZE TO { 1.000000000E0,
                                        4.072265600E-01,
                                        -4.09176900E-01 } )
 %GIP( C_Single_2 : FLOAT ARRAY (3) INITIALIZE TO { 1.000000000E0,
```

```
                                                 1.746093750E-01,
                                                 -8.466796900E-01 } )
%GIP( C_Single_3 : FLOAT ARRAY (3) INITIALIZE TO { 1.000000000E0,
                                                 9.306640600E-01,
                                                 -3.027343800E-01 } )
%GIP( Eckart_Taps_1 : FLOAT ARRAY(3) INITIALIZE TO { 1.000000000E0,
                                                   1.996093750E-01,
                                                   1.000000000E0 } )
%GIP( Eckart_Taps_2 : FLOAT ARRAY(3) INITIALIZE TO { 1.000000000E0,
                                                   1.894531250E-01,
                                                   1.000000000E0 } )
%GIP( Eckart_Taps_3 : FLOAT ARRAY(3) INITIALIZE TO { 1.000000000E0,
                                                   0.000000000E0,
                                                   -1.000000000E0 } )
%QUEUE( Beam_filt_data : FLOAT )
%QUEUE( STA_Beam_rep : FLOAT )
%QUEUE( STA_Beam : FLOAT )
%QUEUE( Sigma : FLOAT ARRAY (1))
%QUEUE( Beam_Equal : FLOAT )
%SUBGRAPH( BDF_Single
   GRAPH    = Eckart
   GIP      = NBMS,
            NAVE,
            NT,
            C_Single_1,
            C_Single_2,
            C_Single_3,
            Eckart_Taps_1,
            Eckart_Taps_2,
            Eckart_Taps_3
   INPUTQ   = Beam_data
   OUTPUTQ  = Beam_filt_data )
%SUBGRAPH( Rec_Ave
   GRAPH    = STA
   GIP      = NAVE,
            NBMS
   INPUTQ   = Beam_filt_data
   OUTPUTQ  = STA_Beam_rep,
            STA_Beam )
%SUBGRAPH( Aperture_Norm
   GRAPH    = Normal
   GIP      = NBMS
   VAR      = Clip_Level
   INPUTQ   = STA_Beam_rep
   OUTPUTQ  = Sigma )
%SUBGRAPH( Equal
   GRAPH    = Aperture_Equal
   GIP      = NBMS
   INPUTQ   = STA_Beam,
            Sigma
   OUTPUTQ  = Beam_Equal )
%SUBGRAPH( Integration
   GRAPH    = Beam_Integration
   GIP      = NBMS,
            NINT
   INPUTQ   = Beam_Equal
   OUTPUTQ  = Beam_out )
%ENDGRAPH
```

Eckart Filter

The Eckart Filter processing is shown in Figure 2. The first action is to transpose the input data. This results in a matrix of `NPTS` (128) rows by `NBEAMS` (55) columns or multiplexed data. This can be useful since there are a number of primitives such as FIR and IIR filters and demodulation routines that act on multiplexed data. Processing the multiplexed data as a unit leads to large data amounts with small data availability latency. By processing large data amounts as a unit, graph execution overhead is kept low.

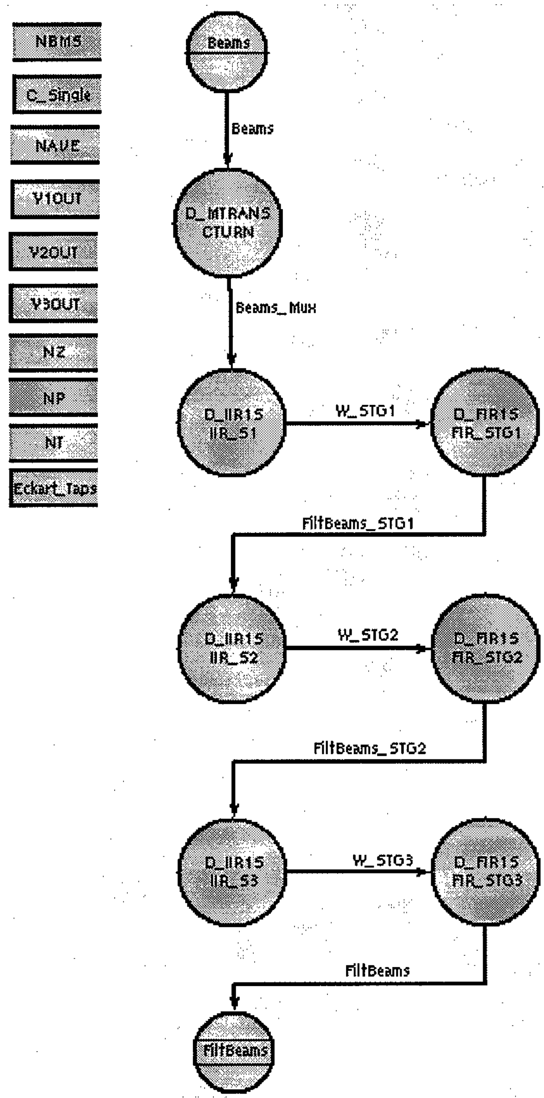

Figure 2. Eckart Filter Processing

After obtaining multiplexed data, the SRS specified Eckart filtering which implemented as three stages of IIR filters. However, the mathematical description described first processing the poles of the filter and then the zeros. The current implementation of the IIR Domain Primitive first does the processing associated with the zeros and then the processing associated with the poles.

Since the pole processing is recursive, the processing is not interchangeable, rather the two implementations are duals. Rather than recalculating the pole and zero coefficients for the dual, it was decided to implement the processing as an IIR filter with no zeros followed by a FIR filter that performed the zero processing. This is reflected in the graph shown in Figure 2.

The SPGN for the Eckart Filter processing is shown below.

```
%GRAPH (ECKART
   GIP = NBMS : INT,
       NAVE : INT,
       NT : INT,
       C_SINGLE_1 : FLOAT ARRAY (3),
       C_SINGLE_2: FLOAT ARRAY (3),
       C_SINGLE_3 : FLOAT ARRAY (3),
       ECKART_TAPS_1 : FLOAT ARRAY (3),
       ECKART_TAPS_2 : FLOAT ARRAY (3),
       ECKART_TAPS_3 : FLOAT ARRAY (3)
   INPUTQ = BEAMS : FLOAT
   OUTPUTQ = FILTBEAMS : FLOAT)

%GIP (NZ : INT
   INITIALIZE TO 0)
%GIP (NP : INT
   INITIALIZE TO 2)
%QUEUE (BEAMS_MUX : FLOAT)
%VAR (Y1OUT : FLOAT ARRAY(110)
   INITIALIZE TO {110 OF 0.00000000000000E+00})
%VAR (Y2OUT : FLOAT ARRAY(110)
   INITIALIZE TO {110 OF 0.00000000000000E+00})
%VAR (Y3OUT : FLOAT ARRAY(110)
   INITIALIZE TO {110 OF 0.00000000000000E+00})
%QUEUE (W_STG1 : FLOAT
   INITIALIZE TO (NBMS * (NT - 1)) OF 0.00000000000000E+00)
%QUEUE (FILTBEAMS_STG1 : FLOAT)
%QUEUE (W_STG2 : FLOAT
   INITIALIZE TO (NBMS * (NT - 1)) OF 0.00000000000000E+00)
%QUEUE (FILTBEAMS_STG2 : FLOAT)
%QUEUE (W_STG3 : FLOAT
   INITIALIZE TO (NBMS * (NT - 1)) OF 0.00000000000000E+00)

%NODE (CTURN
   PRIMITIVE = D_MTRANS
   PRIM_IN =
      NBMS,
      NAVE,
      BEAMS
       THRESHOLD = (NBMS * NAVE)
   PRIM_OUT = BEAMS_MUX)
%NODE (IIR_S1
   PRIMITIVE = D_IIR1S
   PRIM_IN =
      NAVE,
      NBMS,
      NZ,
      NP,
      1,
```

```
        C_SINGLE_1,
        0,
        BEAMS_MUX
         THRESHOLD = NBMS*NAVE,
        Y1OUT
     PRIM_OUT =
        W_STG1,
        Y1OUT)
  %NODE (IIR_S2
     PRIMITIVE = D_IIR1S
     PRIM_IN =
        NAVE,
        NBMS,
        NZ,
        NP,
        1,
        C_SINGLE_2,
        0,
        FILTBEAMS_STG1
         THRESHOLD = NBMS*NAVE,
        Y2OUT
     PRIM_OUT =
        W_STG2,
        Y2OUT)
  %NODE (IIR_S3
     PRIMITIVE = D_IIR1S
     PRIM_IN =
        NAVE,
        NBMS,
        NZ,
        NP,
        1,
        C_SINGLE_3,
        0,
        FILTBEAMS_STG2
         THRESHOLD = NAVE*NBMS,
        Y3OUT
     PRIM_OUT =
        W_STG3,
        Y3OUT)
  %NODE (FIR_STG1
     PRIMITIVE = D_FIR1S
     PRIM_IN =
        NAVE+NT-1,
        NBMS,
        NT,
        1,
        ECKART_TAPS_1,
        W_STG1
         THRESHOLD = ((NAVE + (NT - 1)) * NBMS)
         %%THRESHOLD = (NAVE((NBMS + (NT - 1))) * NAVE)
         CONSUME = (NBMS * NAVE)
     PRIM_OUT = FILTBEAMS_STG1)
  %NODE (FIR_STG2
     PRIMITIVE = D_FIR1S
     PRIM_IN =
        NAVE+NT-1,
        NBMS,
```

```
        NT,
        1,
        ECKART_TAPS_2,
        W_STG2
         THRESHOLD = ((NAVE + (NT - 1)) * NBMS)
         CONSUME = (NAVE * NBMS)
     PRIM_OUT = FILTBEAMS_STG2)
  %NODE (FIR_STG3
     PRIMITIVE = D_FIR1S
     PRIM_IN =
        NAVE+NT-1,
        NBMS,
        NT,
        1,
        ECKART_TAPS_3,
        W_STG3
         THRESHOLD = ((NAVE + (NT - 1)) * NBMS)
         CONSUME = (NAVE * NBMS)
     PRIM_OUT = FILTBEAMS)
  %ENDGRAPH
```

Short Term Average

The Short Term Average processing consists of converting the beam signals to power and then summing the power over the time length NDATA (128) samples. This processing is shown in Figure 3.

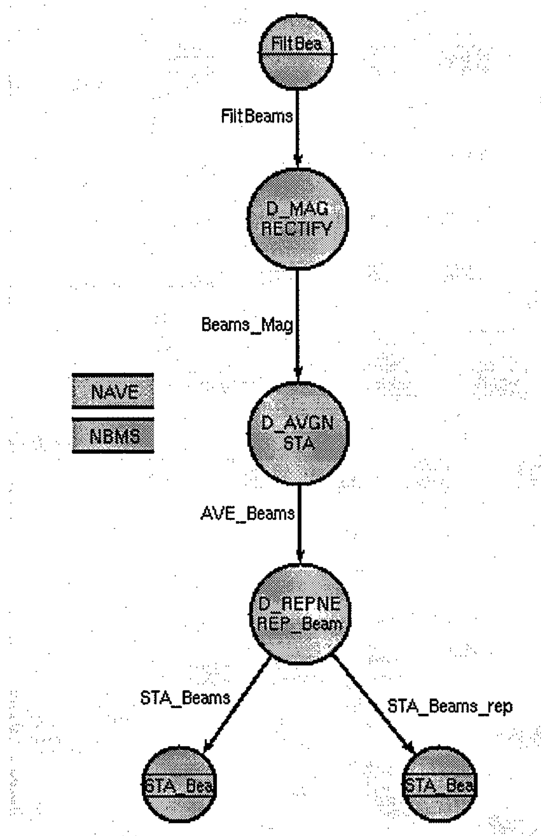

Figure 3. Short Term Average Processing

The SPGN for the Short Term Average processing is shown below.

```
%GRAPH (STA
   GIP = NAVE : INT,
      NBMS : INT
   INPUTQ = FILTBEAMS : FLOAT
   OUTPUTQ = STA_BEAMS : FLOAT,
      STA_BEAMS_REP : FLOAT)
%QUEUE (AVE_BEAMS : FLOAT)
%QUEUE (BEAMS_MAG : FLOAT)
%NODE (STA_NODE
   PRIMITIVE = D_AVGN
   PRIM_IN =
      NBMS,
      NAVE,
      UNUSED,
      UNUSED,
      UNUSED,
      BEAMS_MAG
       THRESHOLD = (NBMS * NAVE)
   PRIM_OUT =
      AVE_BEAMS,
      UNUSED,
      UNUSED)
%NODE (REP_BEAMS
   PRIMITIVE = D_REPNE
   PRIM_IN =
      NBMS,
      2,
      UNUSED,
      AVE_BEAMS
       THRESHOLD = NBMS
   PRIM_OUT = FAMILY [STA_BEAMS, STA_BEAMS_REP])
%NODE (RECTIFY
   PRIMITIVE = D_MAG
   PRIM_IN =
      (NAVE * NBMS),
      FILTBEAMS
       THRESHOLD = (NAVE * NBMS)
   PRIM_OUT = BEAMS_MAG)
%ENDGRAPH
```

Normalizer

The purpose of the Normalizer is to obtain an estimate of the noise floor for all beams of the array. The Normalizer processing is shown in Figure 4. The signal is first clipped at a threshold value of `CLIP` and then averaged over all the beams. The average is then scaled to form a new value of `CLIP`. The average is sent to the Aperture Equalization processing.

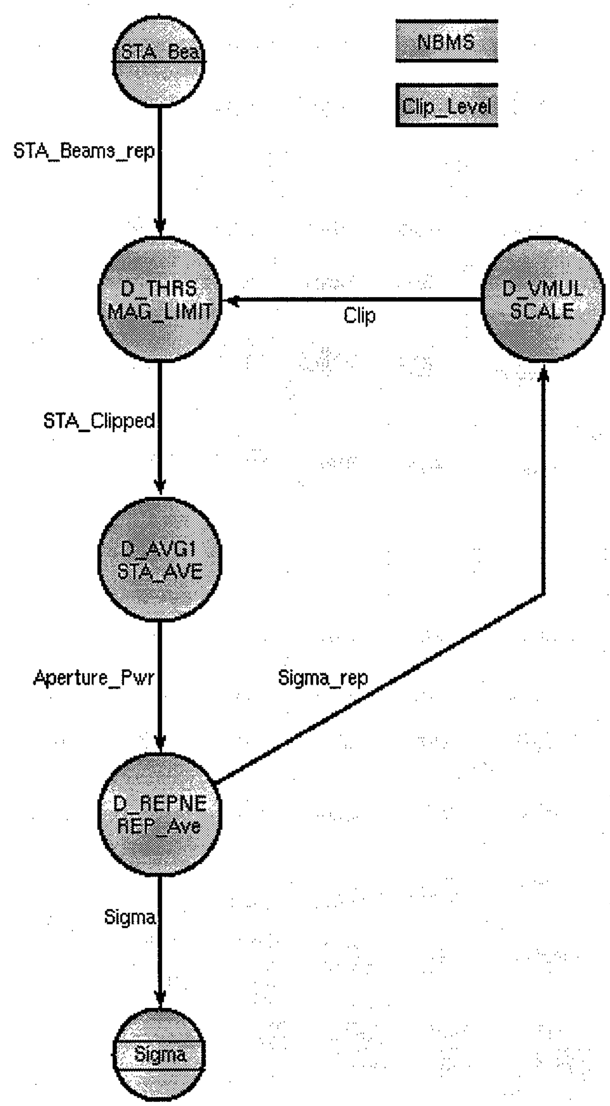

Figure 4. Normalizer Processing

The SPGN for the Normalizer processing is shown below.

```
%GRAPH (NORMAL
   GIP = NBMS : INT
   VAR = CLIP_LEVEL : FLOAT
   INPUTQ = STA_BEAMS_REP : FLOAT
   OUTPUTQ = SIGMA : FLOAT ARRAY (1))
%QUEUE (CLIP : FLOAT
   INITIALIZE TO 2 OF 0.50000000000000E+00)
%QUEUE (STA_CLIPPED : FLOAT)
%QUEUE (APERTURE_PWR : FLOAT)
%QUEUE (SIGMA_REP : FLOAT)
%NODE (MAG_LIMIT
   PRIMITIVE = D_THRS
   PRIM_IN =
      NBMS,
      0,
```

```
      CLIP
       THRESHOLD = 2
       READ      = 1
       OFFSET    = 1
       CONSUME   = 1,
      UNUSED,
      STA_BEAMS_REP
       THRESHOLD = NBMS
   PRIM_OUT = STA_CLIPPED)
%NODE (STA_AVE
   PRIMITIVE = D_AVG1
   PRIM_IN =
      NBMS,
      1,
      STA_CLIPPED
       THRESHOLD = NBMS
   PRIM_OUT = APERTURE_PWR)
%NODE (REP_AVE
   PRIMITIVE = D_REPNE
   PRIM_IN =
      1,
      2,
      UNUSED,
      APERTURE_PWR
       THRESHOLD = 1
   PRIM_OUT = FAMILY [SIGMA, SIGMA_REP])
%NODE (SCALE
   PRIMITIVE = D_VMUL
   PRIM_IN =
      1,
      UNUSED,
      CLIP_LEVEL,
      SIGMA_REP
       THRESHOLD = 1
   PRIM_OUT = CLIP)
%ENDGRAPH
```

Aperture Equalization

The Aperture Equalization processing divides the short term average for each beam by the “noise” average for all beams. This processing is shown in Figure 5.

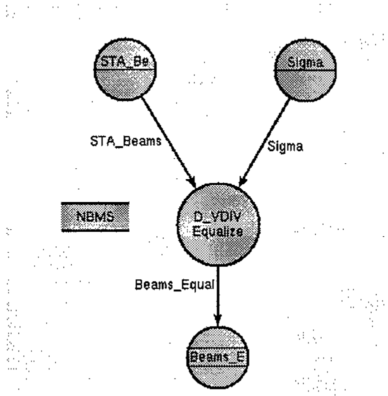

Figure 5. Aperture Equalization Processing

The SPGN for the Aperture Equalization processing is shown below.

```
%GRAPH (APERTURE_EQUAL
   GIP = NBMS : INT
   INPUTQ = STA_BEAMS : FLOAT,
          SIGMA : FLOAT ARRAY (1)
   OUTPUTQ = BEAMS_EQUAL : FLOAT)
%NODE (EQUALIZE
   PRIMITIVE = D_VDIV
   PRIM_IN =
      NBMS,
      STA_BEAMS
       THRESHOLD = NBMS,
      SIGMA
       THRESHOLD = 1
   PRIM_OUT = BEAMS_EQUAL)
%ENDGRAPH
```

Beam Integration

The Beam Integration processing averages NINT (set to 8) samples of the equalized short term averaged data for each beam. This processing is shown in Figure 6.

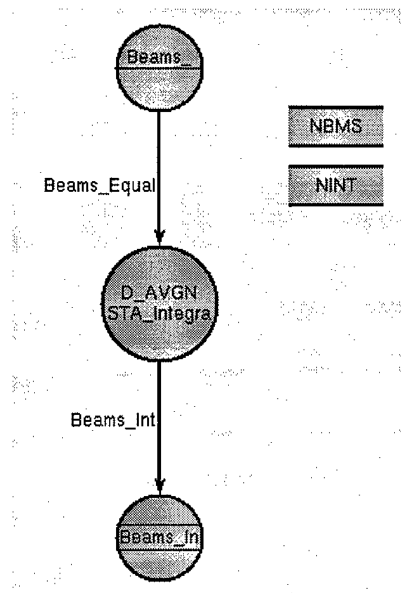

Figure 6. Beam Integration Processing

The SPGN for the Beam Integration processing is shown below.

```
%GRAPH (BEAM_INTEGRATION
   GIP = NBS : INT,
      NINT : INT
   INPUTQ = BEAMS_EQUAL : FLOAT
   OUTPUTQ = BEAMS_INT : FLOAT)
%NODE (STA_INTEGRATE
   PRIMITIVE = D_AVGN
   PRIM_IN =
      NBS,
      NINT,
```

```
        UNUSED,
        UNUSED,
        UNUSED,
        BEAMS_EQUAL
         THRESHOLD = (NINT * NBS)
     PRIM_OUT = BEAMS_INT,
              UNUSED,
              UNUSED)
%ENDGRAPH
```

Simulated Input

The simulated data input to the Broadband Array processing represented time domain output from the beamformer. Beam patterns were simulated as shown in Figure 7. Beam gain was obtained by using linear interpolation based on the direction of the target from the beam steering direction. Each "target" was represented with a target bearing, a target bearing rate, and a target strength. The signal from each target was implemented as pseudo random broadband noise at a level corresponding to the target strength. If no target signal was within a beam, the beam was given a broadband noise signal.

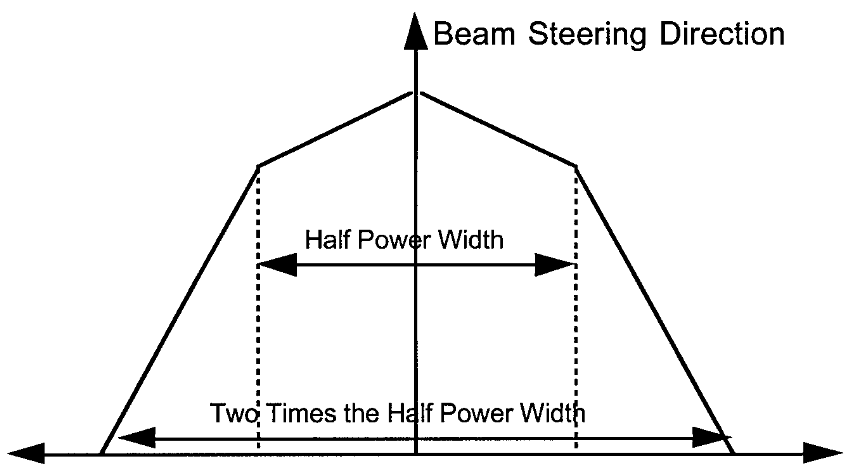

Figure 7. Beam Gain Approximation Used for Simulated Data

Output Display

A typical output display is shown in Figure 8 for a single target that has a high bearing rate. The display is a waterfall type display with the X axis indicating the bearing from 0 to 180 degrees. Each beam is represented as a band of pixels. Each beam is nominally three degrees.

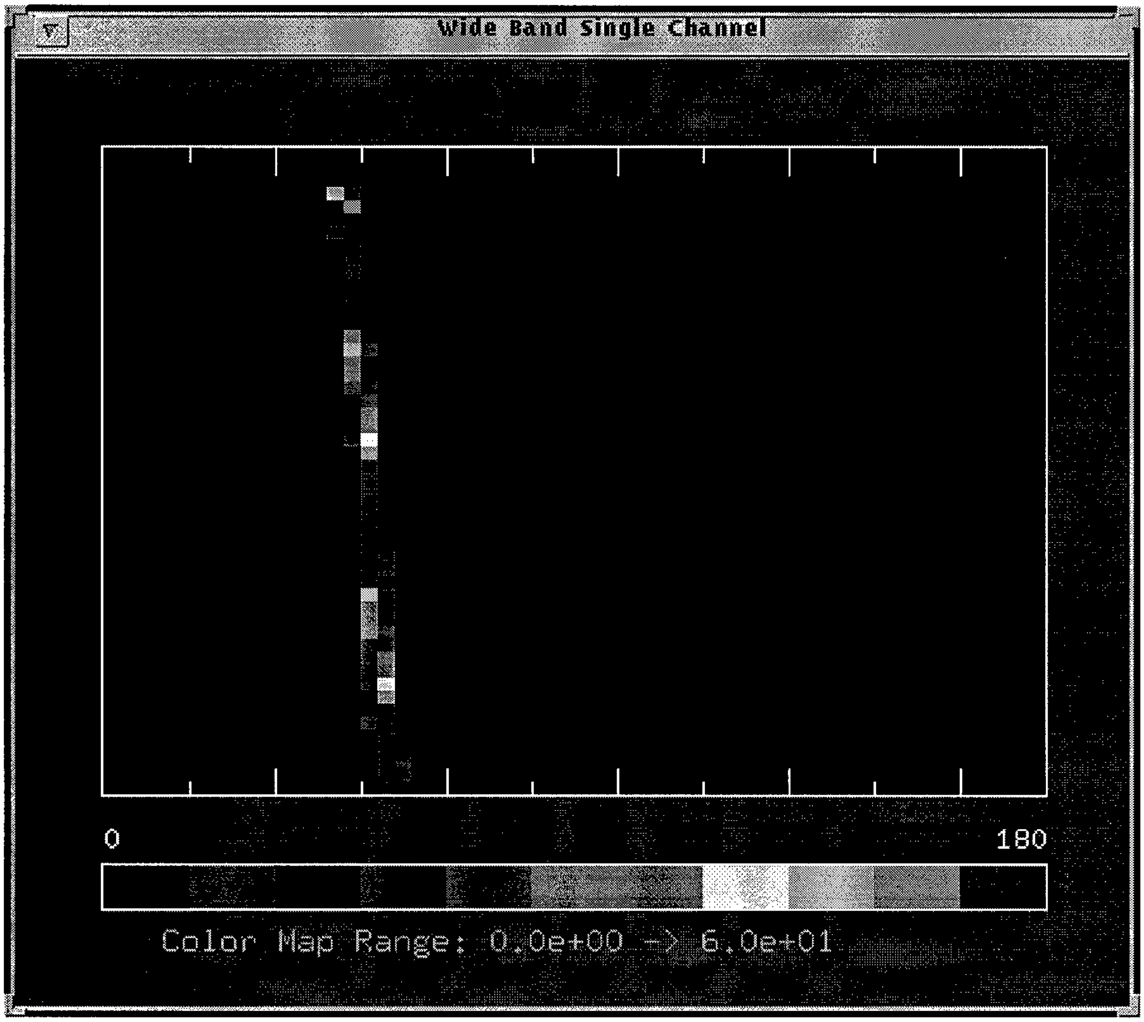

Figure 8. Typical Display For Single Target with Bearing Rate

Broadband Array Pair

Overview of the Processing

The Broadband Array Pair processing performs the Broadband Array processing on the beamformed output from two arrays and then combines the results.

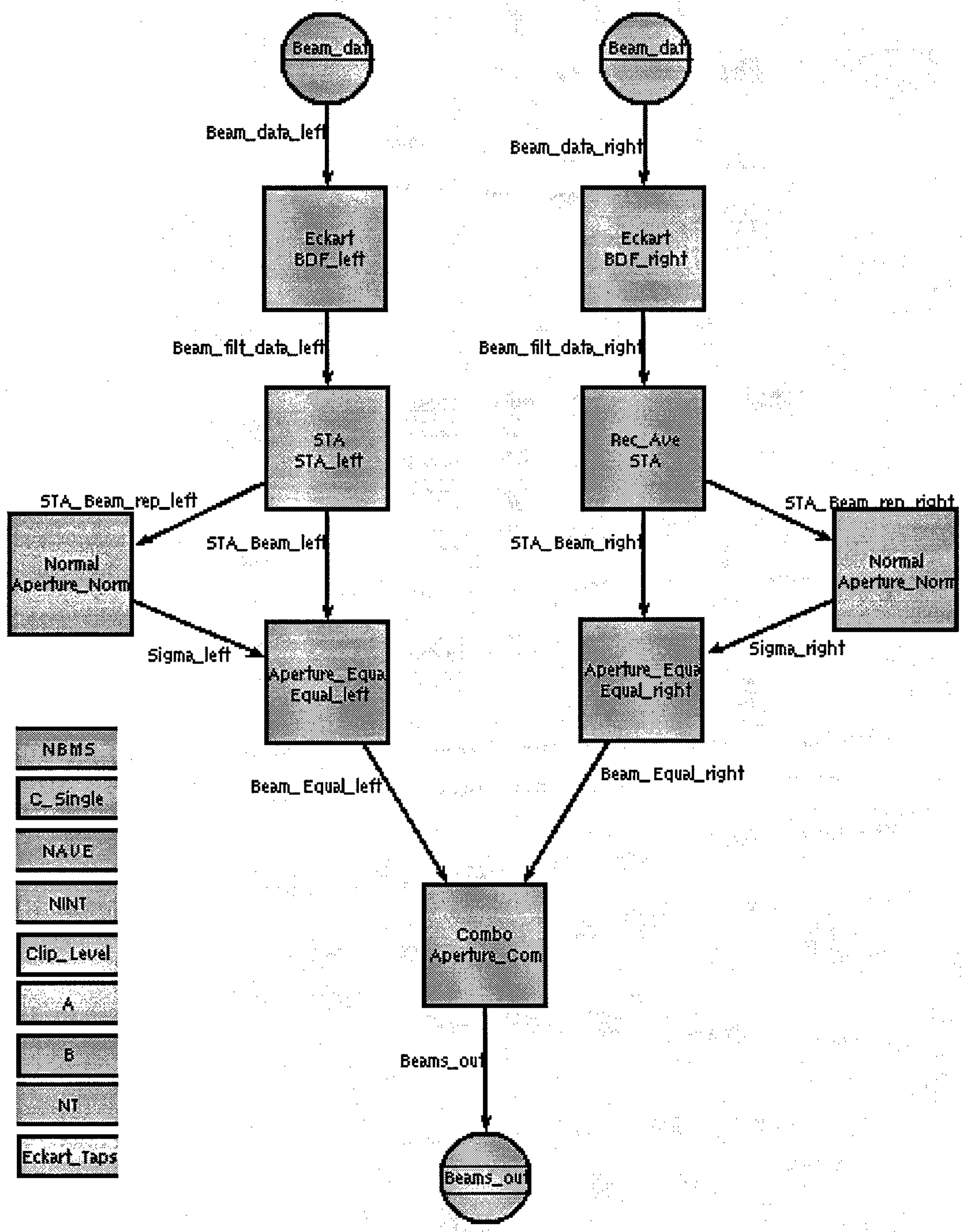

Figure 9. Broadband Array Pair Processing

The SPGN for the Broadband Array Pair processing is shown below.

```
%GRAPH (LRP
   VAR     = C_Single_1 : FLOAT ARRAY (3),
             C_Single_2 : FLOAT ARRAY (3),
             C_Single_3 : FLOAT ARRAY (3),
             Clip_Level : FLOAT,
             A : FLOAT ARRAY (1),
             B : FLOAT ARRAY (1),
             Eckart_Taps_1 : FLOAT ARRAY(3),
             Eckart_Taps_2 : FLOAT ARRAY(3),
             Eckart_Taps_3 : FLOAT ARRAY(3)
   INPUTQ = BEAM_DATA_RIGHT : FLOAT,
      BEAM_DATA_LEFT : FLOAT
   OUTPUTQ = BEAMS_OUT : FLOAT)
%GIP (NBMS : INT
   INITIALIZE TO 55)
%GIP (NAVE : INT
   INITIALIZE TO 128)
%GIP (NINT : INT
   INITIALIZE TO 8)
%GIP (NT : INT
   INITIALIZE TO 3)
%QUEUE (BEAM_FILT_DATA_RIGHT : FLOAT)
%QUEUE (STA_BEAM_REP_RIGHT : FLOAT)
%QUEUE (STA_BEAM_RIGHT : FLOAT)
%QUEUE (SIGMA_RIGHT : FLOAT ARRAY (1))
%QUEUE (BEAM_EQUAL_RIGHT : FLOAT)
%QUEUE (BEAM_FILT_DATA_LEFT : FLOAT)
%QUEUE (STA_BEAM_REP_LEFT : FLOAT)
%QUEUE (STA_BEAM_LEFT : FLOAT)
%QUEUE (SIGMA_LEFT : FLOAT ARRAY (1))
%QUEUE (BEAM_EQUAL_LEFT : FLOAT)
%SUBGRAPH (BDF_RIGHT
   GRAPH = ECKART
   GIP = NBMS, NAVE, NT
   VAR = C_SINGLE_1,
         C_SINGLE_2,
         C_SINGLE_3,
         ECKART_TAPS_1,
         ECKART_TAPS_2,
         ECKART_TAPS_3
   INPUTQ = BEAM_DATA_RIGHT
   OUTPUTQ = BEAM_FILT_DATA_RIGHT)
%SUBGRAPH (STA_RIGHT
   GRAPH = STA
   GIP = NAVE, NBMS
   INPUTQ = BEAM_FILT_DATA_RIGHT
   OUTPUTQ = STA_BEAM_REP_RIGHT, STA_BEAM_RIGHT)
%SUBGRAPH (APERTURE_NORMAL_RIGHT
   GRAPH = NORMAL
   GIP = NBMS
   VAR = CLIP_LEVEL
   INPUTQ = STA_BEAM_REP_RIGHT
   OUTPUTQ = SIGMA_RIGHT)
%SUBGRAPH (EQUAL_RIGHT
   GRAPH = APERTURE_EQUAL
   GIP = NBMS
```

```
    INPUTQ = STA_BEAM_RIGHT, SIGMA_RIGHT
    OUTPUTQ = BEAM_EQUAL_RIGHT)
%SUBGRAPH (BDF_LEFT
    GRAPH = ECKART
    GIP = NBMS, NAVE, NT
    VAR = C_SINGLE_1,
          C_SINGLE_2,
          C_SINGLE_3,
          ECKART_TAPS_1,
          ECKART_TAPS_2,
          ECKART_TAPS_3
    INPUTQ = BEAM_DATA_LEFT
    OUTPUTQ = BEAM_FILT_DATA_LEFT)
%SUBGRAPH (STA_LEFT
    GRAPH = STA
    GIP = NAVE, NBMS
    INPUTQ = BEAM_FILT_DATA_LEFT
    OUTPUTQ = STA_BEAM_REP_LEFT, STA_BEAM_LEFT)
%SUBGRAPH (APERTURE_NORMAL_LEFT
    GRAPH = NORMAL
    GIP = NBMS
    VAR = CLIP_LEVEL
    INPUTQ = STA_BEAM_REP_LEFT
    OUTPUTQ = SIGMA_LEFT)
%SUBGRAPH (EQUAL_LEFT
    GRAPH = APERTURE_EQUAL
    GIP = NBMS
    INPUTQ = STA_BEAM_LEFT, SIGMA_LEFT
    OUTPUTQ = BEAM_EQUAL_LEFT)
%SUBGRAPH (APERTURE_COMBINE
    GRAPH = COMBO
    GIP = NBMS,
          NINT
    VAR = A,
          B
    INPUTQ = BEAM_EQUAL_LEFT, BEAM_EQUAL_RIGHT
    OUTPUTQ = BEAMS_OUT)
%ENDGRAPH
```

Eckart Filter

This processing is the same as the Broadband Array Eckart Filter processing.

Short Term Average

This processing is the same as the Broadband Array Short Term Average processing.

Normalizer

This processing is the same as the Broadband Array Normalizer processing.

Aperture Equalization

This processing is the same as the Broadband Array Aperture Equalization processing.

Aperture Combine

The Aperture Combine processing scales the aperture equalized output for each array and then sums the scaled values. The processing is shown in Figure 10.

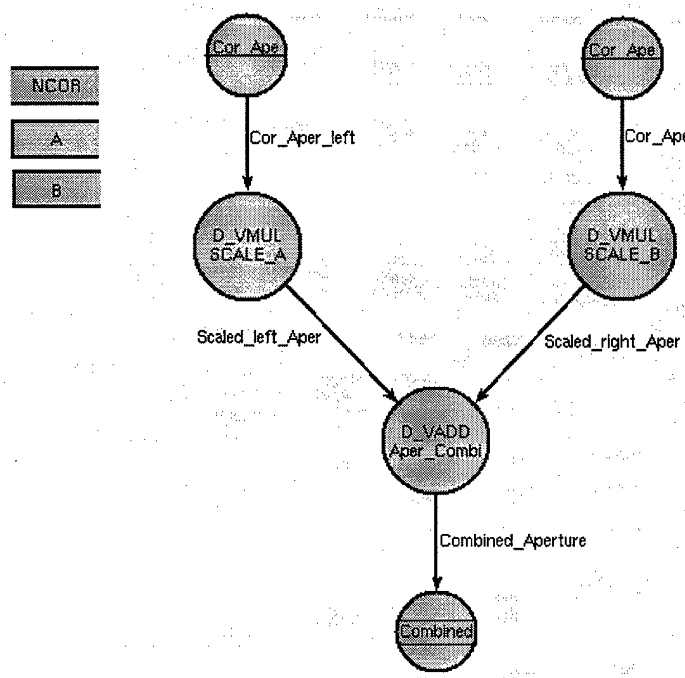

Figure 10. Aperture Combine Processing

The SPGN for the Aperture Combine processing is shown below.

```
%GRAPH (COMBO
   GIP = NBMS : INT,
      NINT : INT
   VAR = A : FLOAT ARRAY (1),
       B : FLOAT ARRAY (1)
   INPUTQ = BEAMS_LEFT : FLOAT,
      BEAMS_RIGHT : FLOAT
   OUTPUTQ = COMBINED_BEAMS : FLOAT)
%QUEUE (INT_BEAMS_LEFT : FLOAT)
%QUEUE (INT_BEAMS_RIGHT : FLOAT)
%QUEUE (SCALED_BEAMS_LEFT : FLOAT)
%QUEUE (SCALED_BEAMS_RIGHT : FLOAT)
%NODE (INTEGRATE_LEFT
   PRIMITIVE = D_AVGN
   PRIM_IN =
      NBMS,
      NINT,
      UNUSED,
      UNUSED,
      UNUSED,
      BEAMS_LEFT
       THRESHOLD = (NINT * NBMS)
```

```
   PRIM_OUT = INT_BEAMS_LEFT,
              UNUSED,
              UNUSED)
%NODE (INTEGRATE_RIGHT
   PRIMITIVE = D_AVGN
   PRIM_IN =
      NBMS,
      NINT,
      UNUSED,
      UNUSED,
      UNUSED,
      BEAMS_RIGHT
       THRESHOLD = (NINT * NBMS)
   PRIM_OUT = INT_BEAMS_RIGHT,
              UNUSED,
              UNUSED)
%NODE (SCALE_LEFT
   PRIMITIVE = D_VMUL
   PRIM_IN =
      NBMS,
      UNUSED,
      INT_BEAMS_LEFT
       THRESHOLD = NBMS,
      A
   PRIM_OUT = SCALED_BEAMS_LEFT)
%NODE (SCALE_RIGHT
   PRIMITIVE = D_VMUL
   PRIM_IN =
      NBMS,
      UNUSED,
      INT_BEAMS_RIGHT
       THRESHOLD = NBMS,
      B
   PRIM_OUT = SCALED_BEAMS_RIGHT)
%NODE (COMBINE
   PRIMITIVE = D_VADD
   PRIM_IN =
      NBMS,
      SCALED_BEAMS_LEFT
       THRESHOLD = NBMS,
      SCALED_BEAMS_RIGHT
       THRESHOLD = NBMS
   PRIM_OUT = COMBINED_BEAMS)
%ENDGRAPH
```

Simulated Input

The simulated input for the Broadband Array Pair processing is identical to the Broadband Array simulated input data except that wideband signals are generated for each array. The same signal was input to each array. The two arrays for each side (right and left) are designed for different frequency bands. Since the signal is wideband, and each processing arm contains filtering to select the appropriate band, using the same signal for both the low and the high frequency array is considered suitable for demonstration purposes. Also, since the processing is independent of time delay, using the same signal for the left and right arrays was considered to sufficient to demonstrate the processing.

Broadband Array Cross-correlation

Overview of the Processing

The Broadband Array Cross-correlation processing consists of filtering, noise estimation and normalization, and three point cross-correlation between beamformed outputs from two arrays referenced to a common origin. The arrays are positioned such that a one sample lead or lag between the two beamformer outputs corresponds to a one degree bearing shift to either the left or the right dependent upon which signal leads the other. The processing is shown in Figure 11.

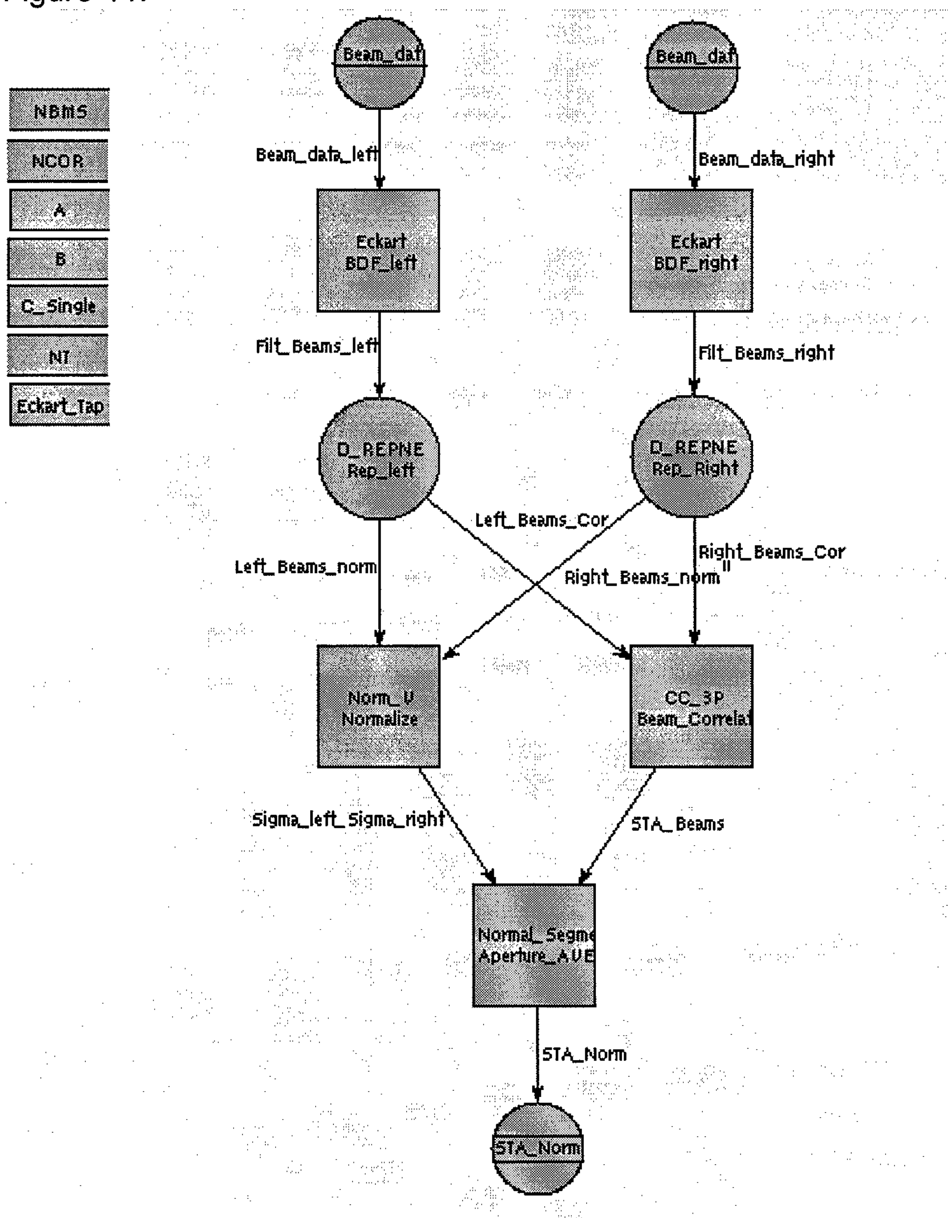

Figure 11. Broadband Array Cross-correlation Processing

The SPGN for the Broadband Array Cross-correlation processing is shown below.

```
%GRAPH (CC3P_S
   VAR = A : FLOAT ARRAY(1),
         B : FLOAT ARRAY(1),
         C_SINGLE_1 : FLOAT ARRAY(3),
         C_SINGLE_2 : FLOAT ARRAY(3),
         C_SINGLE_3 : FLOAT ARRAY(3),
         ECKART_TAPS_1 : FLOAT ARRAY(3),
         ECKART_TAPS_2 : FLOAT ARRAY(3),
         ECKART_TAPS_3 : FLOAT ARRAY(3)
   INPUTQ = BEAM_DATA_RIGHT : FLOAT,
      BEAM_DATA_LEFT : FLOAT
   OUTPUTQ = STA_NORM : FLOAT)

%GIP (NBMS : INT
   INITIALIZE TO 55)
%GIP (NCOR : INT
   INITIALIZE TO 128)
%GIP (NT : INT
   INITIALIZE TO 3)
%QUEUE (FILT_BEAMS_LEFT : FLOAT)
%QUEUE (FILT_BEAMS_RIGHT : FLOAT)
%QUEUE (LEFT_BEAMS_COR : FLOAT)
%QUEUE (RIGHT_BEAMS_COR : FLOAT)
%QUEUE (LEFT_BEAMS_NORM : FLOAT)
%QUEUE (RIGHT_BEAMS_NORM : FLOAT)
%QUEUE (SIGMA_LEFT_SIGMA_RIGHT : FLOAT)
%QUEUE (STA_BEAMS : FLOAT)

%SUBGRAPH (BDF_RIGHT
   GRAPH = ECKART
   GIP = NBMS, NCOR, NT
   VAR = C_SINGLE_1,
         C_SINGLE_2,
         C_SINGLE_3,
         ECKART_TAPS_1,
         ECKART_TAPS_2,
         ECKART_TAPS_3
   INPUTQ = BEAM_DATA_RIGHT
   OUTPUTQ = FILT_BEAMS_RIGHT)
%SUBGRAPH (BDF_LEFT
   GRAPH = ECKART
   GIP = NBMS, NCOR, NT
   VAR = C_SINGLE_1,
         C_SINGLE_2,
         C_SINGLE_3,
         ECKART_TAPS_1,
         ECKART_TAPS_2,
         ECKART_TAPS_3
   INPUTQ = BEAM_DATA_LEFT
   OUTPUTQ = FILT_BEAMS_LEFT)
%NODE (REP_LEFT
   PRIMITIVE = D_REPNE
   PRIM_IN =
      (NBMS * NCOR),
      2,
      UNUSED,
```

```
        FILT_BEAMS_LEFT
            THRESHOLD = (NBMS * NCOR)
    PRIM_OUT = FAMILY [LEFT_BEAMS_COR, LEFT_BEAMS_NORM])
%NODE (REP_RIGHT
    PRIMITIVE = D_REPNE
    PRIM_IN =
        (NBMS * NCOR),
        2,
        UNUSED,
        FILT_BEAMS_RIGHT
            THRESHOLD = (NBMS * NCOR)
    PRIM_OUT = FAMILY [RIGHT_BEAMS_COR, RIGHT_BEAMS_NORM])
%SUBGRAPH (BEAM_CORRELATE
    GRAPH = CC_3P
    GIP = NCOR, NBMS
    INPUTQ = LEFT_BEAMS_COR, RIGHT_BEAMS_COR
    OUTPUTQ = STA_BEAMS)
%SUBGRAPH (NORMALIZE
    GRAPH = NORM_V
    GIP = NCOR, NBMS
    INPUTQ = LEFT_BEAMS_NORM, RIGHT_BEAMS_NORM
    OUTPUTQ = SIGMA_LEFT_SIGMA_RIGHT)
%SUBGRAPH (APERTURE_AVE
    GRAPH = NORM_SEGMENT
    GIP = NBMS
    INPUTQ = SIGMA_LEFT_SIGMA_RIGHT, STA_BEAMS
    OUTPUTQ = STA_NORM)
%ENDGRAPH
```

Eckart Filter

This processing is the same as the Broadband Array Eckart Filter processing.

Three Point Cross-correlation

The Three Point Cross-correlation processing is shown in Figure 12. The data from the right beam is copied into three data streams, each with a different time delay. The "right lead" data stream is delayed by two data points, the "right center" data stream is delayed by one data point and the "right lag" data stream has no delay. The data from the left array is delayed by one data sample, thus synchronizing it with the "right center" data stream. The beams from both arrays are transposed in order to demultiplex the data. This has the effect of placing the time samples for each beam into contiguous memory locations. The transposed data from the left array is then copied into three data streams. Cross-correlation is then performed by calculating the inner product on the processed data streams from the left and right arrays. This calculation is repeated for each beam. The output is then formatted such that the lead, same, and lag cross-correlations for each beam are placed into the output data stream.

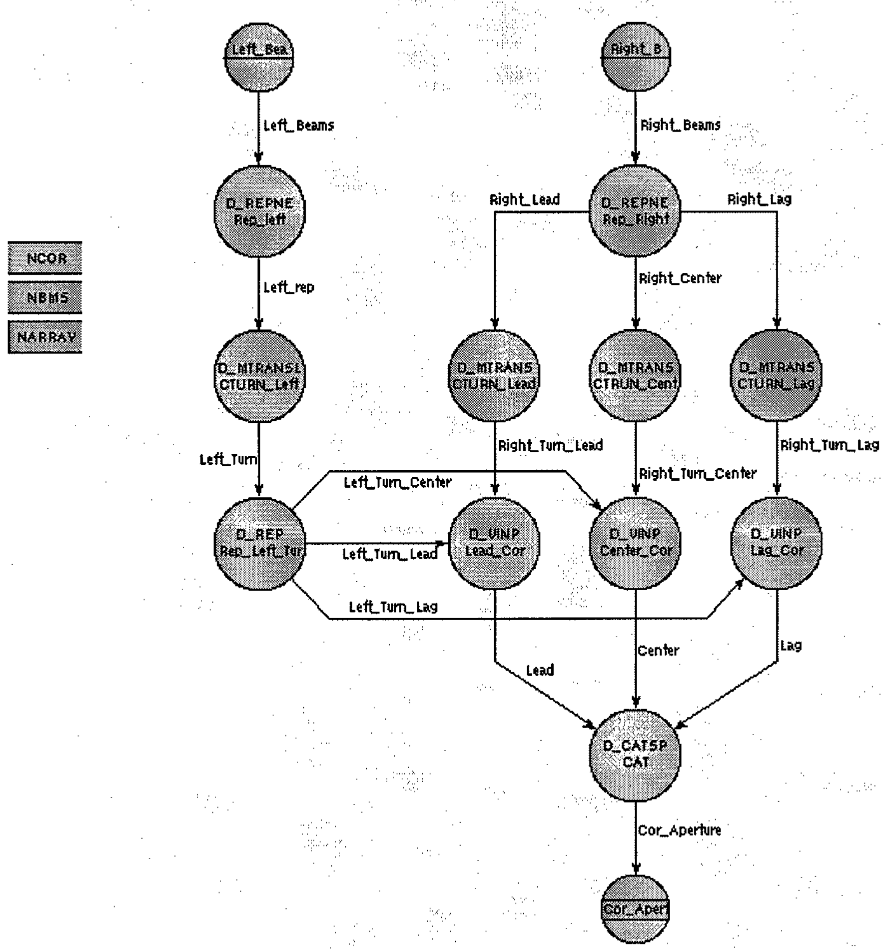

Figure 12. Three Point Cross-correlation Processing

The SPGN for the Three Point Cross-correlation processing is shown below.

```
%GRAPH (CC_3P
   GIP = NCOR, NBMS : INT
   INPUTQ = LEFT_BEAMS : FLOAT,
      RIGHT_BEAMS : FLOAT
   OUTPUTQ = COR_APERTURE : FLOAT)
%QUEUE (RIGHT_LEAD : FLOAT
   INITIALIZE TO (2 * NBMS) OF 0.00000000000000E+00)
%QUEUE (RIGHT_CENTER : FLOAT
   INITIALIZE TO NBMS OF 0.00000000000000E+00)
%QUEUE (RIGHT_LAG : FLOAT)
%QUEUE (LEFT_REP : FLOAT
   INITIALIZE TO NBMS OF 0.00000000000000E+00)
%QUEUE (LEFT_TURN : FLOAT)
%QUEUE (RIGHT_TURN_LEAD : FLOAT)
%QUEUE (LEFT_TURN_LEAD : FLOAT)
%QUEUE (RIGHT_TURN_CENTER : FLOAT)
%QUEUE (LEFT_TURN_CENTER : FLOAT)
%QUEUE (RIGHT_TURN_LAG : FLOAT)
%QUEUE (LEFT_TURN_LAG : FLOAT)
%QUEUE (LAG : FLOAT)
%QUEUE (CENTER : FLOAT)
```

```
%QUEUE (LEAD : FLOAT)
%GIP (NARRAY : INT ARRAY(3)
   INITIALIZE TO {3 OF 1})
%NODE (REP_LEFT
   PRIMITIVE = D_REPNE
   PRIM_IN =
      (NCOR * NBMS),
      1,
      UNUSED,
      LEFT_BEAMS
       THRESHOLD = (NCOR * NBMS)
   PRIM_OUT = FAMILY [LEFT_REP])
%NODE (REP_RIGHT
   PRIMITIVE = D_REPNE
   PRIM_IN =
      (NBMS * NCOR),
      3,
      UNUSED,
      RIGHT_BEAMS
       THRESHOLD = (NCOR * NBMS)
   PRIM_OUT = FAMILY [RIGHT_LEAD, RIGHT_CENTER, RIGHT_LAG])
%NODE (CTURN_LEFT
   PRIMITIVE = D_MTRANS
   PRIM_IN =
      NCOR,
      NBMS,
      LEFT_REP
       THRESHOLD = ((NCOR + 1) * NBMS)
       READ = (NCOR * NBMS)
       CONSUME = (NCOR * NBMS)
   PRIM_OUT = LEFT_TURN)
%NODE (CTURN_LEAD
   PRIMITIVE = D_MTRANS
   PRIM_IN =
      NCOR,
      NBMS,
      RIGHT_LEAD
       THRESHOLD = ((NCOR + 2) * NBMS)
       READ = (NCOR * NBMS)
       CONSUME = (NCOR * NBMS)
   PRIM_OUT = RIGHT_TURN_LEAD)
%NODE (CTRUN_CENTER
   PRIMITIVE = D_MTRANS
   PRIM_IN =
      NCOR,
      NBMS,
      RIGHT_CENTER
       THRESHOLD = ((NCOR + 1) * NBMS)
       READ = (NCOR * NBMS)
       CONSUME = (NCOR * NBMS)
   PRIM_OUT = RIGHT_TURN_CENTER)
%NODE (CTURN_LAG
   PRIMITIVE = D_MTRANS
   PRIM_IN =
      NCOR,
      NBMS,
      RIGHT_LAG
       THRESHOLD = (NCOR * NBMS)
```

```
   PRIM_OUT = RIGHT_TURN_LAG)
%NODE (REP_LEFT_TURN
   PRIMITIVE = D_REP
   PRIM_IN =
      (NBMS * NCOR),
      3,
      LEFT_TURN
       THRESHOLD = (NBMS * NCOR)
   PRIM_OUT = FAMILY [LEFT_TURN_LEAD, LEFT_TURN_CENTER, LEFT_TURN_LAG])
%NODE (LEAD_COR
   PRIMITIVE = D_VINP
   PRIM_IN =
      NCOR,
      RIGHT_TURN_LEAD
       THRESHOLD = (NBMS * NCOR),
      LEFT_TURN_LEAD
       THRESHOLD = (NBMS * NCOR)
   PRIM_OUT = LEAD)
%NODE (CENTER_COR
   PRIMITIVE = D_VINP
   PRIM_IN =
      NCOR,
      RIGHT_TURN_CENTER
       THRESHOLD = (NBMS * NCOR),
      LEFT_TURN_CENTER
       THRESHOLD = (NBMS * NCOR)
   PRIM_OUT = CENTER)
%NODE (LAG_COR
   PRIMITIVE = D_VINP
   PRIM_IN =
      NCOR,
      RIGHT_TURN_LAG
       THRESHOLD = (NBMS * NCOR),
      LEFT_TURN_LAG
       THRESHOLD = (NBMS * NCOR)
   PRIM_OUT = LAG)
%NODE (CAT
   PRIMITIVE = D_CATSP
   PRIM_IN =
      3,
      1,
      NARRAY,
      0,
      NBMS,
      FAMILY [LEAD, CENTER, LAG]
       THRESHOLD = NBMS
   PRIM_OUT = COR_APERTURE)
%ENDGRAPH
```

Normalization

The Normalization processing (NORM_V) consists of converting the filtered data streams from each array into power, averaging over time for the NDATA time length, then further averaging over 16 data sets. This processing is performed for each beam, thus providing an estimate of the energy received in each beam. If perfect correlation is achieved, the value of the correlated signals will be approximately equal to this power. If no correlation is achieved, the value of the

correlated signal will be much smaller than this power. This processing is shown in Figure 13.

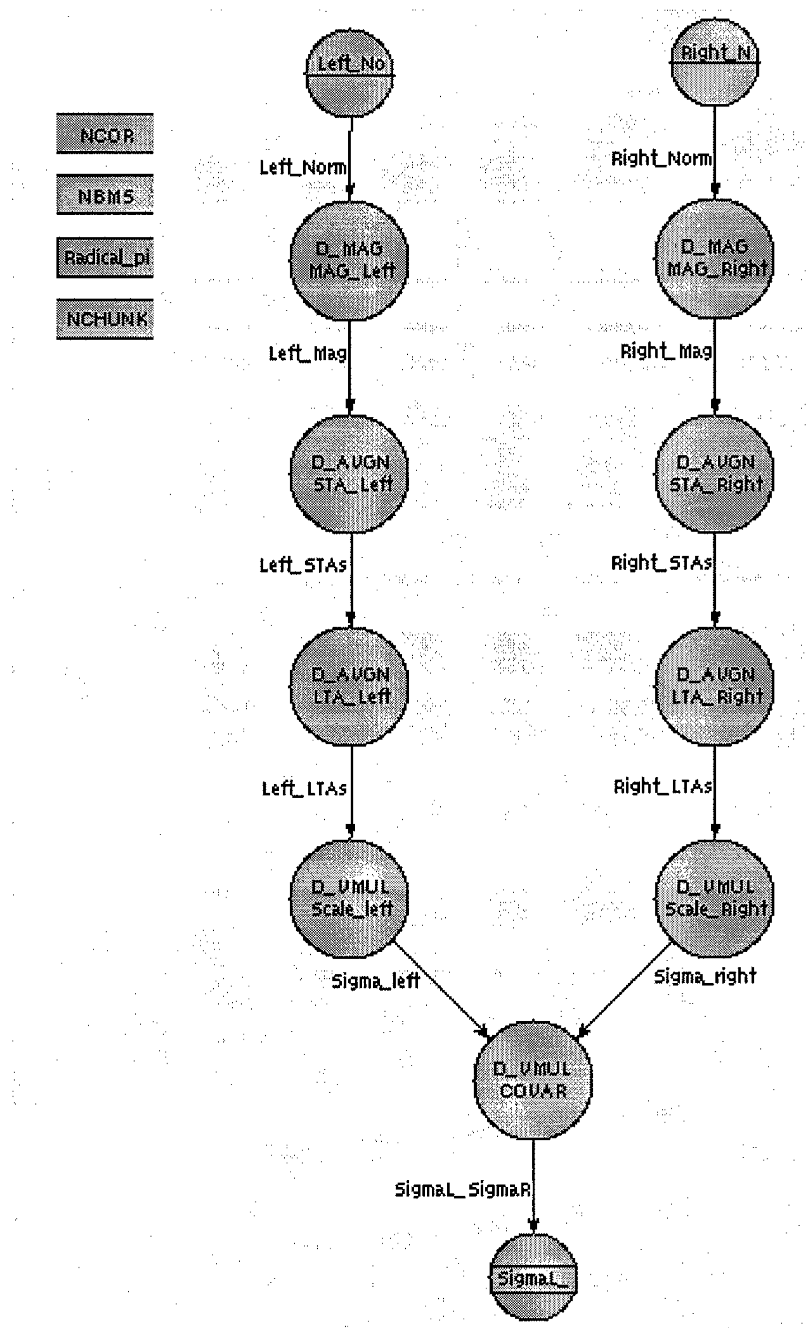

Figure 13. Normalization (Norm_v) Processing

The SPGN for the Normalization (Norm_v) processing is shown below.

```
%GRAPH (NORM_V
   GIP = NCOR : INT,
      NBMS : INT
   INPUTQ = LEFT_NORM : FLOAT,
```

```
      RIGHT_NORM : FLOAT
   OUTPUTQ = SIGMAL_SIGMAR : FLOAT)
%GIP (NCHUNKS : INT
   INITIALIZE TO 16)
%QUEUE (LEFT_MAG : FLOAT
   INITIALIZE TO NBMS OF 0.00000000000000E+00)
%QUEUE (RIGHT_MAG : FLOAT
   INITIALIZE TO NBMS OF 0.00000000000000E+00)
%QUEUE (LEFT_STAS : FLOAT
   INITIALIZE TO ((NCHUNKS - 1) * NBMS) OF 0.00000000000000E+00)
%QUEUE (RIGHT_STAS : FLOAT
   INITIALIZE TO ((NCHUNKS - 1) * NBMS) OF 0.00000000000000E+00)
%QUEUE (LEFT_LTAS : FLOAT)
%VAR (RADICAL_PIOVRTWO : FLOAT ARRAY (1)
   INITIALIZE TO {1.25331413700000E+00})
%QUEUE (RIGHT_LTAS : FLOAT)
%QUEUE (SIGMA_LEFT : FLOAT)
%QUEUE (SIGMA_RIGHT : FLOAT)
%NODE (MAG_LEFT
   PRIMITIVE = D_MAG
   PRIM_IN =
      (NCOR * NBMS),
      LEFT_NORM
       THRESHOLD = (NCOR * NBMS)
   PRIM_OUT = LEFT_MAG)
%NODE (MAG_RIGHT
   PRIMITIVE = D_MAG
   PRIM_IN =
      (NCOR * NBMS),
      RIGHT_NORM
       THRESHOLD = (NCOR * NBMS)
   PRIM_OUT = RIGHT_MAG)
%NODE (STA_LEFT
   PRIMITIVE = D_AVGN
   PRIM_IN =
      NBMS,
      NCOR,
      UNUSED,
      UNUSED,
      UNUSED,
      LEFT_MAG
       THRESHOLD = ((NCOR + 1) * NBMS)
       READ = (NCOR * NBMS)
       CONSUME = (NCOR * NBMS)
   PRIM_OUT =
      LEFT_STAS,
      UNUSED,
      UNUSED)
%NODE (STA_RIGHT
   PRIMITIVE = D_AVGN
   PRIM_IN =
      NBMS,
      NCOR,
      UNUSED,
      UNUSED,
      UNUSED,
      RIGHT_MAG
       THRESHOLD = ((NCOR + 1) * NBMS)
```

```
        READ = (NCOR * NBMS)
        CONSUME = (NCOR * NBMS)
    PRIM_OUT =
       RIGHT_STAS,
       UNUSED,
       UNUSED)
%NODE (LTA_LEFT
    PRIMITIVE = D_AVGN
    PRIM_IN =
       NBMS,
       NCHUNKS,
       UNUSED,
       UNUSED,
       UNUSED,
       LEFT_STAS
        THRESHOLD = (NCHUNKS * NBMS)
        CONSUME = NBMS
    PRIM_OUT =
       LEFT_LTAS,
       UNUSED,
       UNUSED)
%NODE (LTA_RIGHT
    PRIMITIVE = D_AVGN
    PRIM_IN =
       NBMS,
       NCHUNKS,
       UNUSED,
       UNUSED,
       UNUSED,
       RIGHT_STAS
        THRESHOLD = (NCHUNKS * NBMS)
        CONSUME = NBMS
    PRIM_OUT =
       RIGHT_LTAS,
       UNUSED,
       UNUSED)
%NODE (SCALE_LEFT
    PRIMITIVE = D_VMUL
    PRIM_IN =
       NBMS,
       UNUSED,
       LEFT_LTAS
        THRESHOLD = NBMS,
       RADICAL_PIOVRTWO
    PRIM_OUT = SIGMA_LEFT)
%NODE (SCALE_RIGHT
    PRIMITIVE = D_VMUL
    PRIM_IN =
       NBMS,
       UNUSED,
       RIGHT_LTAS
        THRESHOLD = NBMS,
       RADICAL_PIOVRTWO
    PRIM_OUT = SIGMA_RIGHT)
%NODE (COVAR
    PRIMITIVE = D_VMUL
    PRIM_IN =
       NBMS,
```

```
        UNUSED,
        SIGMA_LEFT
         THRESHOLD = NBMS,
        SIGMA_RIGHT
         THRESHOLD = NBMS
    PRIM_OUT = SIGMAL_SIGMAR)
%ENDGRAPH
```

Aperture Averaging

The Aperture Averaging processing (Norm_seg) is shown in Figure 14. For each beam, the three point correlation values are divided by the covariance obtained from the Normalization (Norm_v) processing.

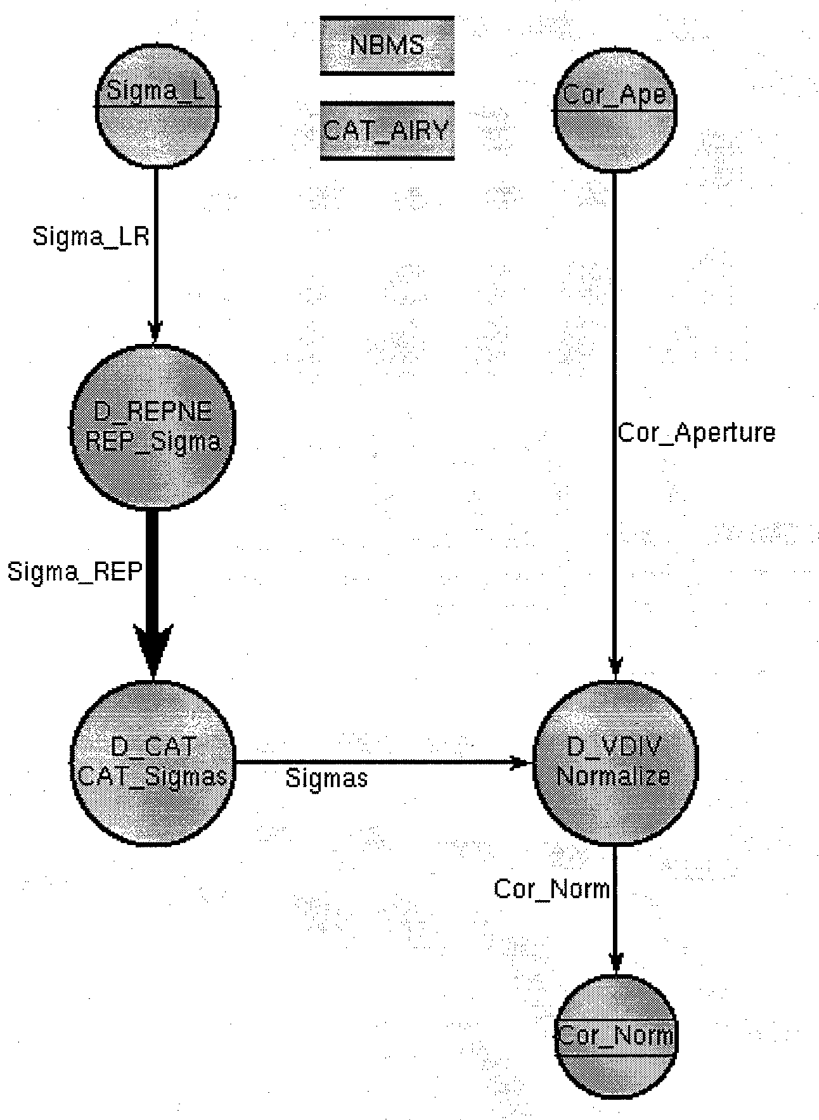

Figure 14. Aperture Averages (Norm_seg)

The SPGN for the Aperture Averages processing (Norm_seg) is shown below.

```
%GRAPH( Norm_Segment
   GIP      = NBMS : INT
   INPUTQ   = Sigma_LR : FLOAT,
            Cor_Aperture : FLOAT
   OUTPUTQ = Cor_Norm : FLOAT )
%QUEUE( [1..3]Sigma_REP : FLOAT )
%QUEUE( Sigmas : FLOAT )
%GIP (CAT_AIRY : INT ARRAY (3) INITIALIZE TO {1, 1, 1})
%NODE( Normalize
   PRIMITIVE = D_VDIV
   PRIM_IN    = 3*NBMS,
            Cor_Aperture THRESHOLD = 3*NBMS,
            Sigmas THRESHOLD = 3*NBMS
   PRIM_OUT  = Cor_Norm )
%NODE( REP_Sigma
   PRIMITIVE = D_REPNE
   PRIM_IN    = NBMS,
            3,
            UNUSED,
            Sigma_LR THRESHOLD = NBMS
   PRIM_OUT  = [1..3]Sigma_REP )
%NODE( CAT_Sigmas
   PRIMITIVE = D_CAT
   PRIM_IN    = 3,
            NBMS,
            CAT_AIRY,
            [1..3]Sigma_REP THRESHOLD = NBMS
   PRIM_OUT  = Sigmas )
%ENDGRAPH
```

Simulated Input

The simulated input is again representative of the output from the beamformer. The same processing was used to calculate beam gain. In this case, time delay is important and the signal generated for one array was either advanced or delayed based on target direction relative to the arrays.

Output Display

A typical output display is shown in Figure 15 for a single target that has a high bearing rate. The display is a waterfall type display with the X axis indicating the bearing from 0 to 180 degrees. Each beam is represented as a band of pixels. Each beam is nominally three degrees; however, the processing resolves the target to one degree. For this processing, there is a rather long transient until only data (rather than zero valued initialization data) is being processed. Much of the transient is captured in the figure.

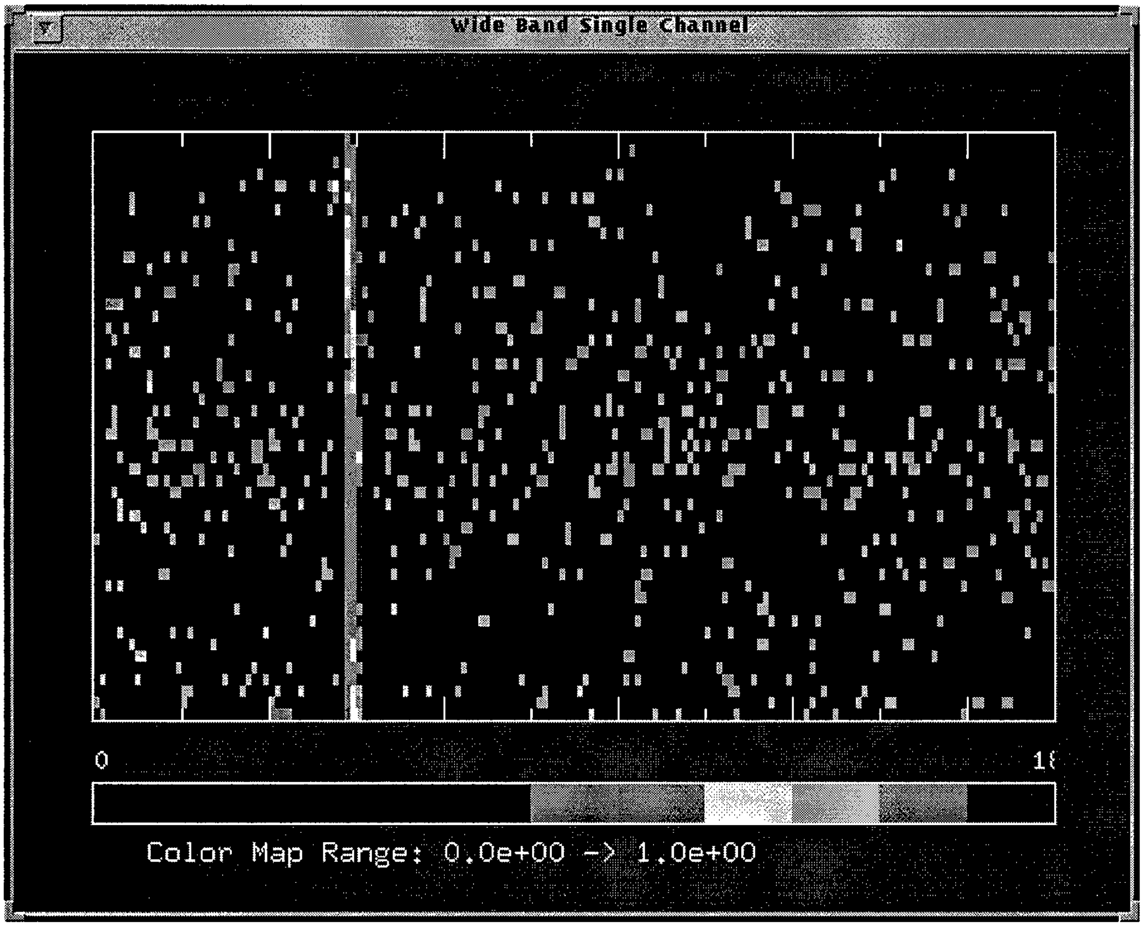

Figure 15. Typical Output Display for Single Target with Bearing Rate

Broadband Array Cross-correlation Pair

Overview of the Processing

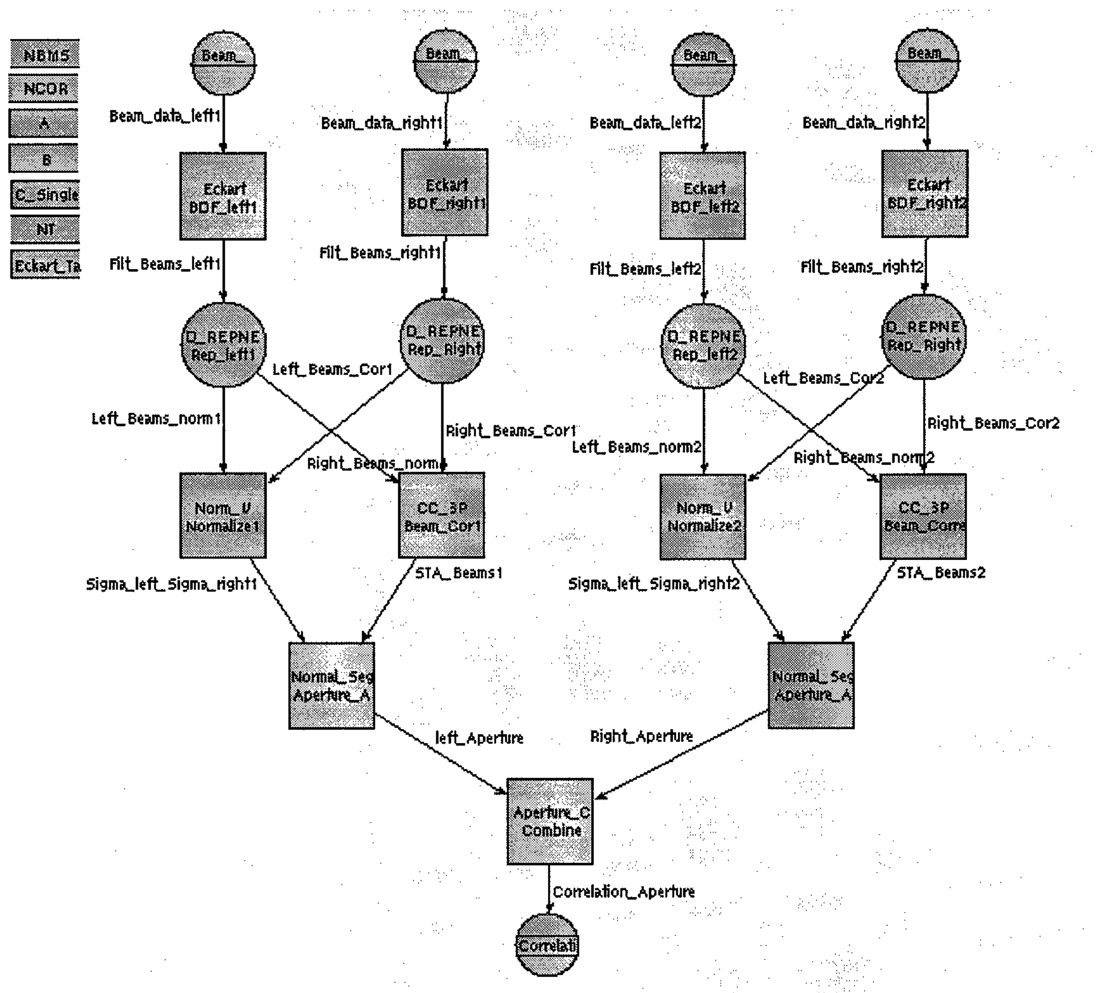

Figure 16. Broadband Array Pair Cross-correlation Processing

The SPGN for the Broadband Array Pair Cross-correlation processing is shown below.

```
%GRAPH (CC3P_P
   VAR = A : FLOAT ARRAY(1),
         B : FLOAT ARRAY(1),
         C_SINGLE_1 : FLOAT ARRAY(3),
         C_SINGLE_2 : FLOAT ARRAY(3),
         C_SINGLE_3 : FLOAT ARRAY(3),
         C_SINGLE_4 : FLOAT ARRAY(3),
         C_SINGLE_5 : FLOAT ARRAY(3),
         C_SINGLE_6 : FLOAT ARRAY(3),
         ECKART_TAPS_1 : FLOAT ARRAY(3),
         ECKART_TAPS_2 : FLOAT ARRAY(3),
         ECKART_TAPS_3 : FLOAT ARRAY(3),
```

```
          ECKART_TAPS_4 : FLOAT ARRAY(3),
          ECKART_TAPS_5 : FLOAT ARRAY(3),
          ECKART_TAPS_6 : FLOAT ARRAY(3)
    INPUTQ = BEAM_DATA_RIGHT1 : FLOAT,
       BEAM_DATA_LEFT1 : FLOAT,
       BEAM_DATA_RIGHT2 : FLOAT,
       BEAM_DATA_LEFT2 : FLOAT
    OUTPUTQ = CORRELATION_APERTURE : FLOAT)
 %GIP (NBMS : INT
    INITIALIZE TO 55)
 %GIP (NCOR : INT
    INITIALIZE TO 128)
 %GIP (NT : INT
    INITIALIZE TO 3)
 %QUEUE (FILT_BEAMS_LEFT1 : FLOAT)
 %QUEUE (FILT_BEAMS_RIGHT1 : FLOAT)
 %QUEUE (LEFT_BEAMS_COR1 : FLOAT)
 %QUEUE (RIGHT_BEAMS_COR1 : FLOAT)
 %QUEUE (LEFT_BEAMS_NORM1 : FLOAT)
 %QUEUE (RIGHT_BEAMS_NORM1 : FLOAT)
 %QUEUE (SIGMA_LEFT_SIGMA_RIGHT1 : FLOAT)
 %QUEUE (STA_BEAMS1 : FLOAT)
 %QUEUE (FILT_BEAMS_LEFT2 : FLOAT)
 %QUEUE (FILT_BEAMS_RIGHT2 : FLOAT)
 %QUEUE (LEFT_BEAMS_COR2 : FLOAT)
 %QUEUE (RIGHT_BEAMS_COR2 : FLOAT)
 %QUEUE (LEFT_BEAMS_NORM2 : FLOAT)
 %QUEUE (RIGHT_BEAMS_NORM2 : FLOAT)
 %QUEUE (SIGMA_LEFT_SIGMA_RIGHT2 : FLOAT)
 %QUEUE (STA_BEAMS2 : FLOAT)
 %QUEUE (LEFT_APERTURE : FLOAT)
 %QUEUE (RIGHT_APERTURE : FLOAT)
 %SUBGRAPH (BDF_RIGHT1
    GRAPH = ECKART
    GIP = NBMS, NCOR, NT
    VAR = C_SINGLE_1,
          C_SINGLE_2,
          C_SINGLE_3,
          ECKART_TAPS_1,
          ECKART_TAPS_2,
          ECKART_TAPS_3
    INPUTQ = BEAM_DATA_RIGHT1
    OUTPUTQ = FILT_BEAMS_RIGHT1)
 %SUBGRAPH (BDF_LEFT1
    GRAPH = ECKART
    GIP = NBMS, NCOR, NT
    VAR = C_SINGLE_1,
          C_SINGLE_2,
          C_SINGLE_3,
          ECKART_TAPS_1,
          ECKART_TAPS_2,
          ECKART_TAPS_3
    INPUTQ = BEAM_DATA_LEFT1
    OUTPUTQ = FILT_BEAMS_LEFT1)
 %NODE (REP_LEFT1
    PRIMITIVE = D_REPNE
    PRIM_IN =
       (NBMS * NCOR),
```

```
      2,
      UNUSED,
      FILT_BEAMS_LEFT1
         THRESHOLD = (NBMS * NCOR)
   PRIM_OUT = FAMILY [LEFT_BEAMS_COR1, LEFT_BEAMS_NORM1])
%NODE (REP_RIGHT1
   PRIMITIVE = D_REPNE
   PRIM_IN =
      (NBMS * NCOR),
      2,
      UNUSED,
      FILT_BEAMS_RIGHT1
         THRESHOLD = (NBMS * NCOR)
   PRIM_OUT = FAMILY [RIGHT_BEAMS_COR1, RIGHT_BEAMS_NORM1])
%SUBGRAPH (BEAM_CORRELATE1
   GRAPH = CC_3P
   GIP = NCOR, NBMS
   INPUTQ = LEFT_BEAMS_COR1, RIGHT_BEAMS_COR1
   OUTPUTQ = STA_BEAMS1)
%SUBGRAPH (NORMALIZE1
   GRAPH = NORM_V
   GIP = NCOR, NBMS
   INPUTQ = LEFT_BEAMS_NORM1, RIGHT_BEAMS_NORM1
   OUTPUTQ = SIGMA_LEFT_SIGMA_RIGHT1)
%SUBGRAPH (APERTURE_AVE1
   GRAPH = NORM_SEGMENT
   GIP = NBMS
   INPUTQ = SIGMA_LEFT_SIGMA_RIGHT1, STA_BEAMS1
   OUTPUTQ = LEFT_APERTURE)
%SUBGRAPH (BDF_RIGHT2
   GRAPH = ECKART
   GIP = NBMS, NCOR, NT
   VAR = C_SINGLE_4,
         C_SINGLE_5,
         C_SINGLE_6,
         ECKART_TAPS_4,
         ECKART_TAPS_5,
         ECKART_TAPS_6
   INPUTQ = BEAM_DATA_RIGHT2
   OUTPUTQ = FILT_BEAMS_RIGHT2)
%SUBGRAPH (BDF_LEFT2
   GRAPH = ECKART
   GIP = NBMS, NCOR, NT
   VAR = C_SINGLE_4,
         C_SINGLE_5,
         C_SINGLE_6,
         ECKART_TAPS_4,
         ECKART_TAPS_5,
         ECKART_TAPS_6
   INPUTQ = BEAM_DATA_LEFT2
   OUTPUTQ = FILT_BEAMS_LEFT2)
%NODE (REP_LEFT2
   PRIMITIVE = D_REPNE
   PRIM_IN =
      (NBMS * NCOR),
      2,
      UNUSED,
      FILT_BEAMS_LEFT2
```

```
            THRESHOLD = (NBMS * NCOR)
      PRIM_OUT = FAMILY [LEFT_BEAMS_COR2, LEFT_BEAMS_NORM2])
   %NODE (REP_RIGHT2
      PRIMITIVE = D_REPNE
      PRIM_IN =
         (NBMS * NCOR),
         2,
         UNUSED,
         FILT_BEAMS_RIGHT2
            THRESHOLD = (NBMS * NCOR)
      PRIM_OUT = FAMILY [RIGHT_BEAMS_COR2, RIGHT_BEAMS_NORM2])
   %SUBGRAPH (BEAM_CORRELATE2
      GRAPH = CC_3P
      GIP = NCOR, NBMS
      INPUTQ = LEFT_BEAMS_COR2, RIGHT_BEAMS_COR2
      OUTPUTQ = STA_BEAMS2)
   %SUBGRAPH (NORMALIZE2
      GRAPH = NORM_V
      GIP = NCOR, NBMS
      INPUTQ = LEFT_BEAMS_NORM2, RIGHT_BEAMS_NORM2
      OUTPUTQ = SIGMA_LEFT_SIGMA_RIGHT2)
   %SUBGRAPH (APERTURE_AVE2
      GRAPH = NORM_SEGMENT
      GIP = NBMS
      INPUTQ = SIGMA_LEFT_SIGMA_RIGHT2, STA_BEAMS2
      OUTPUTQ = RIGHT_APERTURE)
   %SUBGRAPH (COMBINE
      GRAPH = APERTURE_COMBINE
      GIP = NBMS
      VAR = A, B
      INPUTQ = LEFT_APERTURE, RIGHT_APERTURE
      OUTPUTQ = CORRELATION_APERTURE)
   %ENDGRAPH
```

Simulated Input

The simulated input for this processing is identical to the simulated input for the Broadband Array Cross-correlation Processing. The difference between the two is that in the pair processing, two arrays are used in each side, the arrays being designed for two different frequency bands. The filtering contained in the processing selects the bands of interest. Since the simulated input is broadband, this is considered sufficient for demonstration purposes. Time delay between the signals must be considered.

Narrowband Baseline

Overview of the Processing

The Narrowband Baseline processing consists of octave filtering to form seven bands, search processing on all seven bands, and threat processing on the five highest frequency bands. Search processing is performed in two stages. First the data is filtered. Secondly the filtered data is Fourier Transformed and then processed. Threat processing also consists of the same two stages; however, the processing in each stage is different than the Search processing. The top level graph is shown in Figure 17. The band definition processing, search filter and spectrum processing and threat filter and spectrum processing are described in separate sections. Because of the inherent widening of beams for lower frequencies, the number of beams processed for the lower octaves is reduced by approximately a factor of two for each octave.

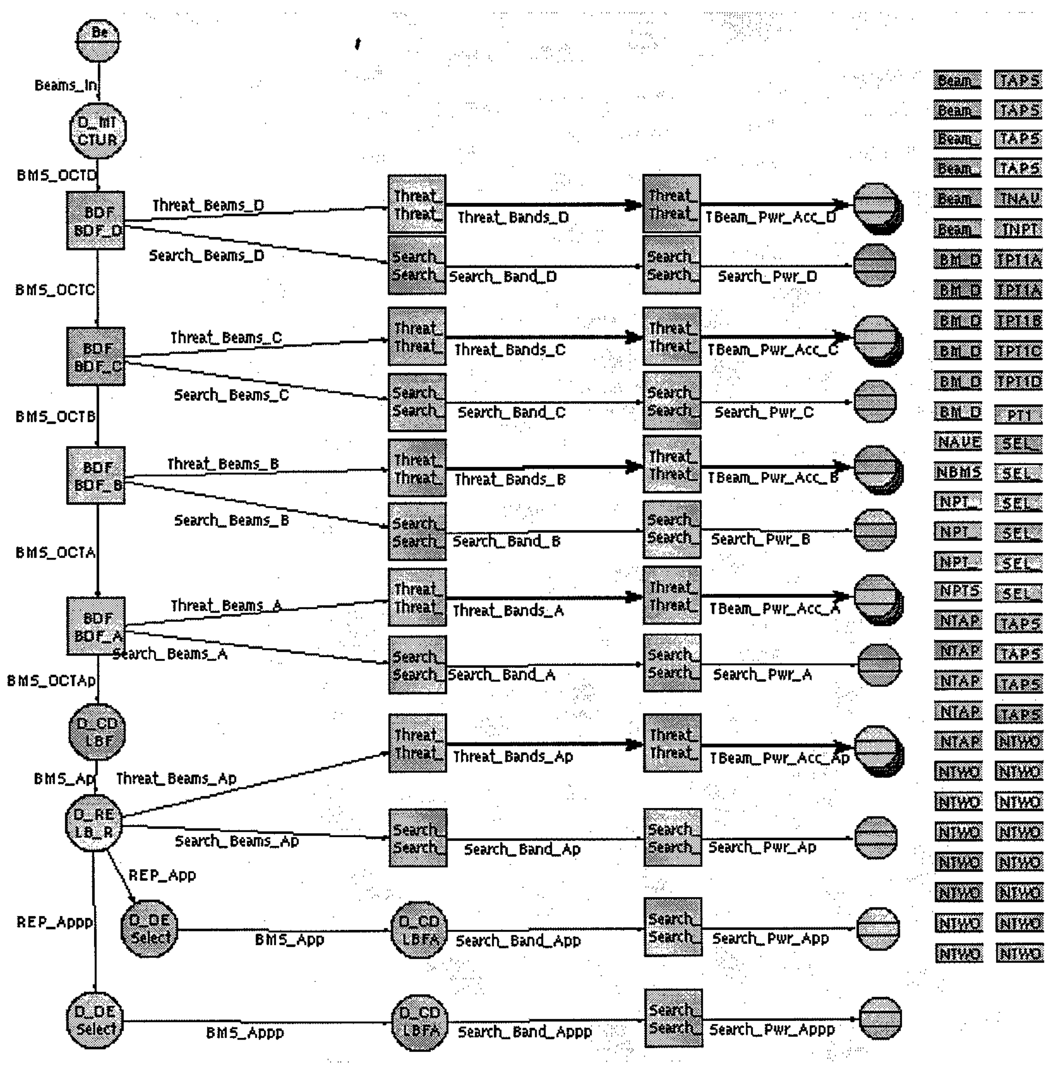

Figure 17. Narrowband Baseline Processing

The SPGN for the Narrowband Baseline processing is:

```
%GRAPH( NB
   GIP      =
              %% Number of FFT output points in threat bands
              TNPTSOUT : INT,
              %% Number of FFT output points in search bands
              NPTSOUT : INT,
              %% Initial output point in search band FFT's
              PT1 : INT,
              %% Initial output point in octave D threat band FFT's
              TPT1D : INT,
              %% Initial output point in octave C threat band FFT's
              TPT1C : INT,
              %% Initial output point in octave B threat band FFT's
              TPT1B : INT,
              %% Initial output point in octave A threat bnad FFT's
              TPT1A : INT,
              %% Initial output point in octave Ap threat band FFT's
              TPT1Ap : INT
   VAR      =
              %% Real filter coefficients for FIR_37
              TAPS_TB8 : FLOAT ARRAY(37),
              %% Real filter coefficients for FIR 75
              TAPS_TB16 : FLOAT ARRAY(75),
              %% Real filter coefficients for FIR_131
              TAPS_TB32 : FLOAT ARRAY(151),
        %% Number of 2 pi periods in demod table for threat filter D:8
        %% octave D
              NTWOPI8D : INT,
        %% Number of 2 pi periods in deomod table of threat filter D:16
        %% octave D
              NTWOPI16D : INT,
        %% Number of 2 pi periods in demod table of threat filter D:32 of
        %% octave D
              NTWOPI32D : INT,
        %% Number of 2 pi periods in demod table of threat filter D:8 of
        %% octave
              NTWOPI8C : INT,
        %% Number of 2 pi periods in demod table of threat filter D:16 of
        %% octave C
              NTWOPI16C : INT,
        %% Number of 2 pi periods in demod table of threat filter D:32 of
        %% octave C
              NTWOPI32C : INT,
        %% Number of 2 pi periods in demod table of threat filter D:8 of
        %% octave B
              NTWOPI8B : INT,
        %% Number of 2 pi periods in demod table of threat filter D:16 of
        %% octave B
              NTWOPI16B : INT,
        %% Number of 2 pi periods in demod table of threat filter D:32 of
        %% octave B
              NTWOPI32B : INT,
        %% Number of 2 pi periods in demod table of threat filter D:8 of
        %% octave A
              NTWOPI8A : INT,
        %% Number of 2 pi periods in demod table of threat filter D:16 of
```

```
        %% octave A
              NTWOPI16A : INT,
        %% Number of 2 pi periods in demod table of threat filter D:32 of
        %% octave A
              NTWOPI32A : INT,
              %% Real filter coefficients for FIR_19
              TAPS_19 : FLOAT ARRAY(19),
              %% Real filter coefficients for FIR_21
              TAPS_21 : FLOAT ARRAY(21),
      %% Number of 2 pi periods in demod table of threat filter D:8 of
      %% octave Ap
              NTWOPI8Ap : INT,
     %% Number of 2 pi periods in demod table of threat filter D:16 of
     %% octave Ap
              NTWOPI16Ap : INT,
     %% Number of 2 pi periods id demod table of threat filter D:32 of
     %% octave Ap
              NTWOPI32Ap : INT,
              %% Real filter coefficients for FIR_11
              TAPS_11 : FLOAT ARRAY(11),
              %% Real filter coefficients for FIR_31
              TAPS_31 : FLOAT ARRAY(31)
    INPUTQ  = Beams_In : FLOAT
    OUTPUTQ = Search_Pwr_D : FLOAT,
              [1..3]TBeam_Pwr_Acc_D : FLOAT,
              Search_Pwr_C : FLOAT,
              [1..3]TBeam_Pwr_Acc_C : FLOAT,
              Search_Pwr_B : FLOAT,
              [1..3]TBeam_Pwr_Acc_B : FLOAT,
              Search_Pwr_A : FLOAT,
              [1..3]TBeam_Pwr_Acc_A : FLOAT,
              Search_Pwr_Ap : FLOAT,
              [1..3]TBeam_Pwr_Acc_Ap : FLOAT,
              Search_Pwr_App : FLOAT,
              Search_Pwr_Appp : FLOAT )
%% Number of input beams
%GIP( NBMS : INT INITIALIZE TO 55 )
%% Beam decimation ratio for octave D
%GIP( BM_DECD : INT INITIALIZE TO 1 )
%% Number of Threat Band PSD's Averaged
%GIP( TNAVE : INT INITIALIZE TO 4 )
%% Number of Search PSD's averaged
%GIP( NAVE : INT INITIALIZE TO 4 )
%% Number of beams selected for octave C
%GIP( SEL_BMSC : INT INITIALIZE TO 55 )
%% Number of beams selected for octave B
%GIP( SEL_BMSB : INT INITIALIZE TO 55 )
%% Number of beams selected for octave A
%GIP( SEL_BMSA : INT INITIALIZE TO 27 )
%% Number of beams selected for octave App
%GIP( SEL_BMSAp : INT INITIALIZE TO 14 )
%% Number of beams selected for octave App
%GIP( SEL_BMSApp : INT INITIALIZE TO 7 )
%% Number of beams selected for octave Appp
%GIP( SEL_BMSAppp : INT INITIALIZE TO 4 )
%% Beam decimation ratio for octave C
%GIP( BM_DECC : INT INITIALIZE TO 1 )
%% Beam decimation ratio for octave B
```

```
%GIP( BM_DECB : INT INITIALIZE TO 2 )
%% Beam decimation ratio for octave A
%GIP( BM_DECA : INT INITIALIZE TO 2 )
%% Beam decimation ratio for octave Ap
%GIP( BM_DECApp : INT INITIALIZE TO 2 )
%% Beam decimation ratio for octave App
%GIP( BM_DECAppp : INT INITIALIZE TO 4 )
%GIP( Beam_offD : INT INITIALIZE TO 0 )
%GIP( Beam_offC : INT INITIALIZE TO 0 )
%GIP( Beam_offB : INT INITIALIZE TO 1 )
%GIP( Beam_offA : INT INITIALIZE TO 0 )
%GIP( Beam_offApp : INT INITIALIZE TO 0 )
%GIP( Beam_offAppp : INT INITIALIZE TO 0 )
%GIP( NTAPS_11 : INT INITIALIZE TO 11 )
%GIP( NTAPS_19 : INT INITIALIZE TO 19 )
%GIP( NTAPS_21 : INT INITIALIZE TO 21 )
%GIP( NTAPS_31 : INT INITIALIZE TO 31 )
%GIP( NTAPS_61 : INT INITIALIZE TO 61 )
%QUEUE( BMS_OCTD : FLOAT )
%QUEUE( Threat_Beams_D : CFLOAT )
%QUEUE( Search_Beams_D : CFLOAT INITIALIZE TO NBMS*(NTAPS_19-2) OF
   <0.0E0,0.0E0> )
%QUEUE( [M=1..3]Threat_Bands_D : CFLOAT )
%QUEUE( BMS_OCTC : FLOAT )
%QUEUE( Threat_Beams_C : CFLOAT )
%QUEUE( Search_Beams_C : CFLOAT INITIALIZE TO SEL_BMSC*(NTAPS_19-2) OF
   <0.0E0,0.0E0> )
%QUEUE( [M=1..3]Threat_Bands_C : CFLOAT )
%QUEUE( BMS_OCTB : FLOAT )
%QUEUE( Threat_Beams_B : CFLOAT )
%QUEUE( Search_Beams_B : CFLOAT INITIALIZE TO SEL_BMSB*(NTAPS_19-2) OF
   <0.0E0,0.0E0> )
%QUEUE( [M=1..3]Threat_Bands_B : CFLOAT )
%QUEUE( Search_Band_B : CFLOAT )
%QUEUE( BMS_OCTA : FLOAT )
%QUEUE( Threat_Beams_A : CFLOAT )
%QUEUE( Search_Beams_A : CFLOAT INITIALIZE TO SEL_BMSA*(NTAPS_19-2) OF
   <0.0E0,0.0E0> )
%QUEUE( [M=1..3]Threat_Bands_A : CFLOAT )
%% Demod table pointer for LBF CDMFIR in octave Ap
%VAR( NPT_LBAp : INT INITIALIZE TO 0 )
%QUEUE( BMS_OCTAp : FLOAT INITIALIZE TO (NTAPS_21-1)*SEL_BMSAp OF
   0.0E0 )
%QUEUE( BMS_Ap : CFLOAT )
%QUEUE( Threat_Beams_Ap : CFLOAT )
%QUEUE( Search_Beams_Ap : CFLOAT INITIALIZE TO SEL_BMSAp*(NTAPS_19-2) OF
   <0.0E0,0.0E0> )
%QUEUE( [M=1..3]Threat_Bands_Ap : CFLOAT )
%QUEUE( Search_Band_Ap : CFLOAT )
%QUEUE( REP_App : CFLOAT )
%QUEUE( REP_Appp : CFLOAT )
%QUEUE( BMS_App : CFLOAT INITIALIZE TO (NTAPS_31-4)*SEL_BMSApp OF
   <0.0E0,0.0E0> )
%QUEUE( BMS_Appp : CFLOAT INITIALIZE TO (NTAPS_61-1)*SEL_BMSAppp OF
   <0.0E0,0.0E0> )
%QUEUE( Search_Band_App : CFLOAT INITIALIZE TO SEL_BMSApp*(NTAPS_19-2) OF
   <0.0E0,0.0E0> )
%QUEUE( Search_Band_Appp : CFLOAT INITIALIZE TO SEL_BMSAppp*(NTAPS_19-2) OF
```

```
   <0.0E0,0.0E0> )
%% Real filter coefficients for FIR_61
%VAR( TAPS_61 : FLOAT ARRAY(61) )
%% Demod table pointer for LBFApp CDMFIR in octave App
%VAR( NPT_LBApp : INT INITIALIZE TO 0 )
%% Demod table pointer for LBFAppp CDMFIR in octave Appp
%VAR( NPT_LBAppp : INT INITIALIZE TO 0 )
%QUEUE( Search_Band_D : CFLOAT )
%QUEUE( Search_Band_C : CFLOAT )
%QUEUE( Search_Band_A : CFLOAT )
%NODE( CTURN
   PRIMITIVE = D_MTRANS
   PRIM_IN   = NBMS,
               3072,
               Beams_In THRESHOLD = NBMS*3072
   PRIM_OUT  = BMS_OCTD )
%SUBGRAPH( BDF_D
   GRAPH    = BDF
   GIP      = NBMS,
              BM_DECD,
              SEL_BMSC,
              Beam_offD
   VAR      = TAPS_11
   INPUTQ   = BMS_OCTD
   OUTPUTQ  = Threat_Beams_D,
              Search_Beams_D,
              BMS_OCTC )
%SUBGRAPH( Threat_D
   GRAPH    = Threat_Filt
   GIP      = NBMS
   VAR      = TAPS_TB8,
              TAPS_TB16,
              TAPS_TB32,
              NTWOPI8D,
              NTWOPI16D,
              NTWOPI32D
   INPUTQ   = Threat_Beams_D
   OUTPUTQ  = [1..3]Threat_Bands_D )
%SUBGRAPH( Search_Filt_D
   GRAPH    = Search_Filt
   GIP      = NBMS
   VAR      = TAPS_19
   INPUTQ   = Search_Beams_D
   OUTPUTQ  = Search_Band_D )
%SUBGRAPH( Threat_Spectrum_D
   GRAPH    = Threat_Spectrum
   GIP      = NBMS,
              TNAVE,
              TNPTSOUT,
              TPT1D
   INPUTQ   = [1..3]Threat_Bands_D
   OUTPUTQ  = [1..3] TBeam_Pwr_Acc_D)
%SUBGRAPH( BDF_C
   GRAPH    = BDF
   GIP      = SEL_BMSC,
              BM_DECC,
              SEL_BMSB,
              Beam_offC
```

```
    VAR     = TAPS_11
    INPUTQ  = BMS_OCTC
    OUTPUTQ = Threat_Beams_C,
              Search_Beams_C,
              BMS_OCTB)
%SUBGRAPH( Threat_C
    GRAPH   = Threat_Filt
    GIP     = SEL_BMSC
    VAR     = TAPS_TB8,
              TAPS_TB16,
              TAPS_TB32,
              NTWOPI8C,
              NTWOPI16C,
              NTWOPI32C
    INPUTQ  = Threat_Beams_C
    OUTPUTQ = [1..3]Threat_Bands_C )
%SUBGRAPH( Search_Filt_C
    GRAPH   = Search_Filt
    GIP     = SEL_BMSC
    VAR     = TAPS_19
    INPUTQ  = Search_Beams_C
    OUTPUTQ = Search_Band_C )
%SUBGRAPH( Threat_Spectrum_C
    GRAPH   = Threat_Spectrum
    GIP     = SEL_BMSC,
              TNAVE,
              TNPTSOUT,
              TPT1B
    INPUTQ  = [1..3]Threat_Bands_C
    OUTPUTQ = [1..3]TBeam_Pwr_Acc_C)
%SUBGRAPH( BDF_B
    GRAPH   = BDF
    GIP     = SEL_BMSB,
              BM_DECB,
              SEL_BMSA,
              Beam_offB
    VAR     = TAPS_11
    INPUTQ  = BMS_OCTB
    OUTPUTQ = Threat_Beams_B,
              Search_Beams_B,
              BMS_OCTA )
%SUBGRAPH( Threat_B
    GRAPH   = Threat_Filt
    GIP     = SEL_BMSB
    VAR     = TAPS_TB8,
              TAPS_TB16,
              TAPS_TB32,
              NTWOPI8B,
              NTWOPI16B,
              NTWOPI32B
    INPUTQ  = Threat_Beams_B
    OUTPUTQ = [1..3]Threat_Bands_B )
%SUBGRAPH( Search_Filt_B
    GRAPH   = Search_Filt
    GIP     = SEL_BMSB
    VAR     = TAPS_19
    INPUTQ  = Search_Beams_B
    OUTPUTQ = Search_Band_B )
```

```
%SUBGRAPH( Threat_Spectrum_B
   GRAPH    = Threat_Spectrum
   GIP      = SEL_BMSB,
              TNAVE,
              TNPTSOUT,
              TPT1C
   INPUTQ   = [1..3]Threat_Bands_B
   OUTPUTQ  = [1..3]TBeam_Pwr_Acc_B)
%SUBGRAPH( Search_Spectrum_B
   GRAPH    = Search_Spectrum
   GIP      = SEL_BMSB,
              NAVE,
              NPTSOUT,
              PT1
   INPUTQ   = Search_Band_B
   OUTPUTQ  = Search_Pwr_B)
%SUBGRAPH( BDF_A
   GRAPH    = BDF
   GIP      = SEL_BMSA,
              BM_DECA,
              SEL_BMSAp,
              Beam_offA
   VAR      = TAPS_11
   INPUTQ   = BMS_OCTA
   OUTPUTQ  = Threat_Beams_A,
              Search_Beams_A,
              BMS_OCTAp )
%SUBGRAPH( Threat_A
   GRAPH    = Threat_Filt
   GIP      = SEL_BMSA
   VAR      = TAPS_TB8,
              TAPS_TB16,
              TAPS_TB32,
              NTWOPI8Ap,
              NTWOPI16Ap,
              NTWOPI32Ap
   INPUTQ   = Threat_Beams_A
   OUTPUTQ  = [1..3]Threat_Bands_A )
%SUBGRAPH( Search_Filt_A
   GRAPH    = Search_Filt
   GIP      = SEL_BMSA
   VAR      = TAPS_19
   INPUTQ   = Search_Beams_A
   OUTPUTQ  = Search_Band_A )
%SUBGRAPH( Threat_Spectrum_A
   GRAPH    = Threat_Spectrum
   GIP      = SEL_BMSA,
              TNAVE,
              TNPTSOUT,
              TPT1A
   INPUTQ   = [1..3]Threat_Bands_A
   OUTPUTQ  = [1..3]TBeam_Pwr_Acc_A)
%NODE( LBF
   PRIMITIVE = D_CDMFIR
   PRIM_IN   = (3072+NTAPS_21)-3,
               SEL_BMSAp,
               0,
               4,
```

```
                1,
                NPT_LBAp,
                NTAPS_21,
                3,
                TAPS_21,
                BMS_OCTAp THRESHOLD = (3072+NTAPS_21 - 3)*SEL_BMSAp
                          CONSUME = 3072*SEL_BMSAp
   PRIM_OUT   = BMS_Ap, NPT_LBAp)
%NODE( LB_REP
   PRIMITIVE = D_REP
   PRIM_IN    = 1024*SEL_BMSAp,
                4,
                BMS_Ap THRESHOLD = 1024 * SEL_BMSAp
   PRIM_OUT   = FAMILY [Threat_Beams_Ap,
                Search_Beams_Ap,
                REP_App,
                REP_Appp] )
%SUBGRAPH( Threat_Ap
   GRAPH     = Threat_Filt
   GIP       = SEL_BMSAp
   VAR       = TAPS_TB8,
               TAPS_TB16,
               TAPS_TB32,
               NTWOPI8Ap,
               NTWOPI16Ap,
               NTWOPI32Ap
   INPUTQ    = Threat_Beams_Ap
   OUTPUTQ = [1..3]Threat_Bands_Ap )
%SUBGRAPH( Search_Filt_Ap
   GRAPH     = Search_Filt
   GIP       = SEL_BMSAp
   VAR       = TAPS_19
   INPUTQ    = Search_Beams_Ap
   OUTPUTQ = Search_Band_Ap )
%SUBGRAPH( Threat_Spectrum_Ap
   GRAPH     = Threat_Spectrum
   GIP       = SEL_BMSAp,
               TNAVE,
               TNPTSOUT,
               TPT1Ap
   INPUTQ    = [1..3]Threat_Bands_Ap
   OUTPUTQ = [1..3]TBeam_Pwr_Acc_Ap)
%SUBGRAPH( Search_Spectrum_Ap
   GRAPH     = Search_Spectrum
   GIP       = SEL_BMSAp,
               NAVE,
               NPTSOUT,
               PT1
   INPUTQ    = Search_Band_Ap
   OUTPUTQ = Search_Pwr_Ap)
%NODE( Select_App
   PRIMITIVE = D_DEC
   PRIM_IN    = SEL_BMSAp - Beam_offApp,
                BM_DECApp,
                REP_App THRESHOLD = 2048*(SEL_BMSAp-Beam_offApp)
   PRIM_OUT   = BMS_App )
%NODE( Select_Appp
   PRIMITIVE = D_DEC
```

```
   PRIM_IN    = SEL_BMSAp - Beam_offAppp,
                BM_DECAppp,
                REP_Appp THRESHOLD = 4096*(SEL_BMSAp - Beam_offAppp)
   PRIM_OUT   = BMS_Appp )
%NODE( LBFApp
   PRIMITIVE = D_CDMFIR
   PRIM_IN    = (2048+NTAPS_31)-4,
                SEL_BMSApp,
                0,
                4,
                1,
                NPT_LBApp,
                NTAPS_31,
                4,
                TAPS_31,
                BMS_App THRESHOLD = ((2048+NTAPS_31)-4)*SEL_BMSApp
                        CONSUME = 2048*SEL_BMSApp
   PRIM_OUT   = Search_Band_App,
                NPT_LBApp )
%NODE( LBFAppp
   PRIMITIVE = D_CDMFIR
   PRIM_IN    = (4096+NTAPS_61)-8,
                SEL_BMSAppp,
                0,
                8,
                3,
                NPT_LBAppp,
                NTAPS_61,
                8,
                TAPS_61,
                BMS_Appp THRESHOLD = ((4096+NTAPS_61)-8)*SEL_BMSAppp
                         CONSUME = 4096*SEL_BMSAppp
   PRIM_OUT   = Search_Band_Appp,
                NPT_LBAppp )
%SUBGRAPH( Search_Spectrum_App
   GRAPH    = Search_Spectrum
   GIP      = SEL_BMSApp,
              NAVE,
              NPTSOUT,
              PT1
   INPUTQ   = Search_Band_App
   OUTPUTQ  = Search_Pwr_App )
%SUBGRAPH( Search_Spectrum_Appp
   GRAPH    = Search_Spectrum
   GIP      = SEL_BMSAppp,
              NAVE,
              NPTSOUT,
              PT1
   INPUTQ   = Search_Band_Appp
   OUTPUTQ  = Search_Pwr_Appp )
%SUBGRAPH( Search_Spectrum_D
   GRAPH    = Search_Spectrum
   GIP      = NBMS,
              NAVE,
              NPTSOUT,
              PT1
   INPUTQ   = Search_Band_D
   OUTPUTQ  = Search_Pwr_D )
```

```
%SUBGRAPH( Search_Spectrum_C
   GRAPH    = Search_Spectrum
   GIP      = SEL_BMSC,
              NAVE,
              NPTSOUT,
              PT1
   INPUTQ   = Search_Band_C
   OUTPUTQ  = Search_Pwr_C )
%SUBGRAPH( Search_Spectrum_A
   GRAPH    = Search_Spectrum
   GIP      = SEL_BMSA,
              NAVE,
              NPTSOUT,
              PT1
   INPUTQ   = Search_Band_A
   OUTPUTQ  = Search_Pwr_A)
%ENDGRAPH
```

Band Definition Filter

The Band Definition Filter processing is shown in Figure 18. The input data is replicated. To form the octave, the data is complex demodulated and filtered using a FIR. This filtered data is replicated with one copy being sent to the search processing and the other copy being sent to Threat processing. The data that is sent to form the subsequent octave formation is decimated and then filtered using a FIR.

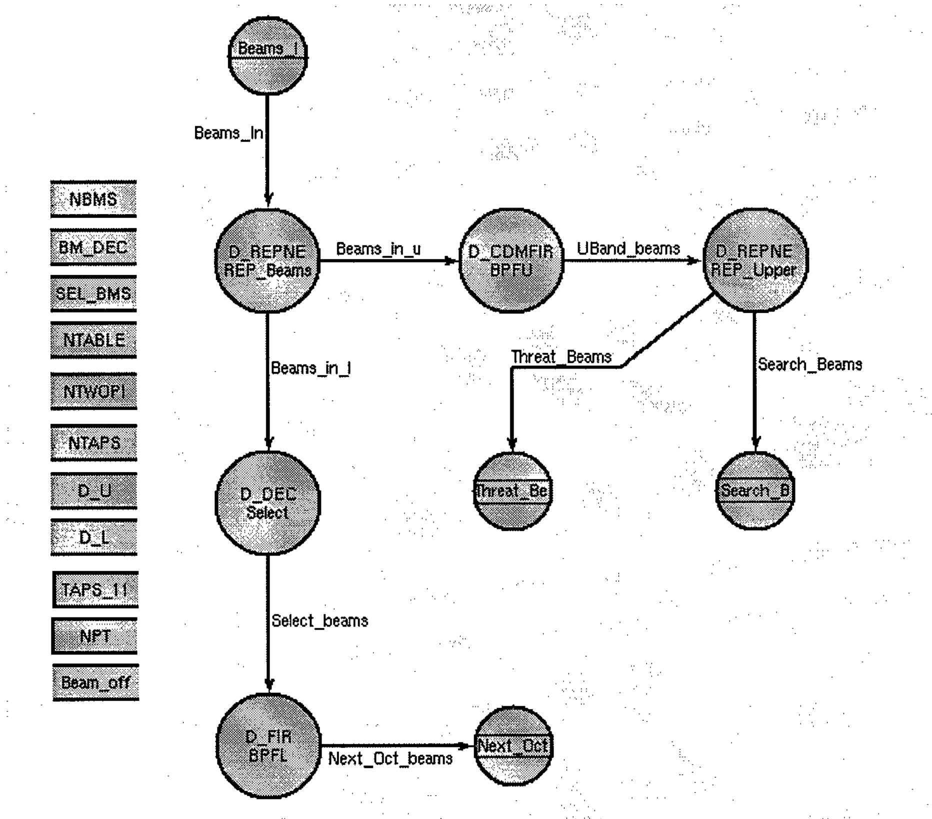

Figure 18. Band Definition Filter Processing

The SPGN for the Band Definition Filter processing is:

```
%GRAPH( BDF
   GIP      = NBMS : INT,
              BM_DEC : INT,
              SEL_BMS : INT,
              Beam_off : INT
   VAR      = TAPS_11 : FLOAT ARRAY(11)
   INPUTQ   = Beams_In : FLOAT
   OUTPUTQ = Threat_Beams : CFLOAT,
              Search_Beams : CFLOAT,
              Next_Oct_beams : FLOAT )
%GIP( NTABLE : INT INITIALIZE TO 52 )
%GIP( NTWOPI : INT INITIALIZE TO 9 )
%GIP( NTAPS : INT INITIALIZE TO 11 )
%GIP( D_U : INT INITIALIZE TO 3 )
%GIP( D_L : INT INITIALIZE TO 2 )
%QUEUE( Beams_in_u : FLOAT INITIALIZE TO (NTAPS-3)*NBMS OF 0.0E0 )
%QUEUE( UBand_beams : CFLOAT )
%QUEUE( Select_beams : FLOAT INITIALIZE TO (NTAPS-2)*SEL_BMS OF
   0.0E0 )
%VAR( NPT : INT INITIALIZE TO 0 )
%QUEUE( Beams_in_l : FLOAT )
%QUEUE( Lopped_BMS : FLOAT )
%NODE( REP_Beams_in
   PRIMITIVE = D_REPNE
   PRIM_IN   = 3072*NBMS,
               2,
               UNUSED,
               Beams_In THRESHOLD = 3072*NBMS
   PRIM_OUT  = FAMILY[Beams_in_u,Beams_in_l] )
%NODE( BPFU
   PRIMITIVE = D_CDMFIR
   PRIM_IN   = (3072+NTAPS)-3,
               NBMS,
               0,
               NTABLE,
               NTWOPI,
               NPT,
               NTAPS,
               D_U,
               TAPS_11,
               Beams_in_u
                   THRESHOLD = ((3072+NTAPS)-3)*NBMS
                   CONSUME = 3072*NBMS
   PRIM_OUT  = UBand_beams,
               NPT )
%NODE( REP_Upper
   PRIMITIVE = D_REPNE
   PRIM_IN   = 1024*NBMS,
               2,
               UNUSED,
               UBand_beams THRESHOLD = 1024*NBMS
   PRIM_OUT  = FAMILY[Threat_Beams,Search_Beams] )
%NODE( Select
   PRIMITIVE = D_DEC
   PRIM_IN   = NBMS - Beam_off,
               BM_DEC,
```

```
                 Lopped_BMS THRESHOLD = (NBMS-Beam_off)*3072
   PRIM_OUT   = Select_beams )
%NODE( BPFL
   PRIMITIVE = D_FIR1S
   PRIM_IN    = (3072+NTAPS)-2,
                 SEL_BMS,
                 NTAPS,
                 D_L,
                 TAPS_11,
                 Select_beams THRESHOLD = ((3072+NTAPS)-2)*SEL_BMS
                              CONSUME = 3072 *SEL_BMS
PRIM_OUT   = Next_Oct_beams )
%NODE( Lop_off
   PRIMITIVE = D_REORD
   PRIM_IN    = NBMS,
                 NBMS-Beam_off,
                 1+Beam_off,
                 NBMS,
                 NBMS,
                 1,
                 Beams_in_1 THRESHOLD = 3072*NBMS
   PRIM_OUT   = Lopped_BMS )
%ENDGRAPH
```

Search Filter

The Search Filter processing is filtering using a FIR filter as shown in Figure 19.

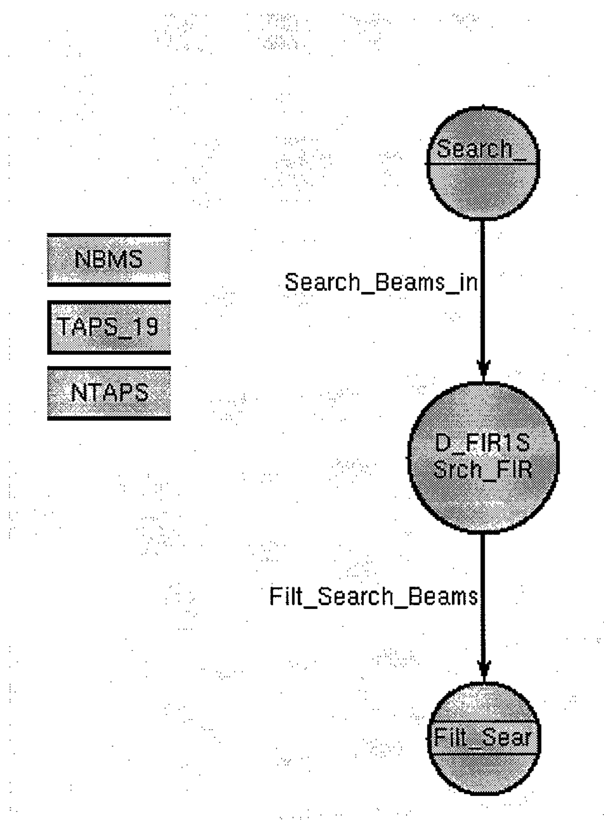

Figure 19. Search Filter Processing

The SPGN for the Search Filter processing is;

```
%GRAPH( SEARCH_FILT
   GIP      = NBMS : INT
   VAR      = TAPS_19 : FLOAT ARRAY(19)
   INPUTQ   = Search_Beams_in : CFLOAT
   OUTPUTQ = Filt_Search_Beams : CFLOAT )
%GIP( NTAPS : INT INITIALIZE TO 19 )
%NODE( Srch_FIR
   PRIMITIVE = D_FIR1S
   PRIM_IN    = (1024+NTAPS)-2,
                 NBMS,
                 NTAPS,
                 2,
                 TAPS_19,
                 Search_Beams_in THRESHOLD = ((1024+NTAPS)-2) * NBMS
                                 CONSUME = 1024 * NBMS
   PRIM_OUT   = Filt_Search_Beams )
%ENDGRAPH
```

Search Spectrum

The Search Spectrum processing, shown in Figure 20, consists of first corner turning the data to demultiplex the data into a time series for each beam, weighting the data with a Hamming function, Fourier transforming the data, and determining the magnitude of the transformed data. This data is then sent to a display. The data is also averaged over the frequency cells to obtain power averages which are sent to two other processing graphs (which are not part of the demonstration).

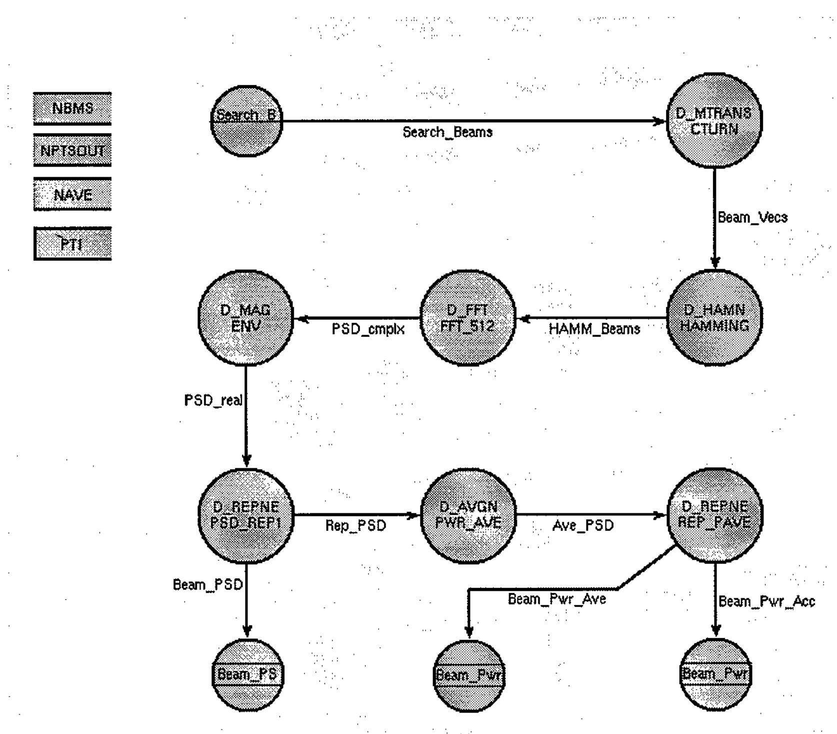

Figure 20. Search Spectrum Processing

The SPGN for the Search Spectrum processing is:

```
%GRAPH( SEARCH_SPECTRUM
   GIP      = NBMS : INT,
              NAVE : INT,
              NPTSOUT : INT,
              PT1 : INT
   INPUTQ  = Search_Beams : CFLOAT
   OUTPUTQ = Beam_Pwr_Acc : FLOAT )
%QUEUE( Beam_Vecs : CFLOAT )
%QUEUE( HAMM_Beams : CFLOAT )
%QUEUE( PSD_cmplx : CFLOAT )
%QUEUE( PSD_real : FLOAT )
%QUEUE( Rep_PSD : FLOAT INITIALIZE TO (NAVE-1)*(NBMS*NPTSOUT) OF
   0.0E0 )
%NODE( CTURN
   PRIMITIVE = D_MTRANS
   PRIM_IN   = 512,
               NBMS,
               Search_Beams THRESHOLD = 512*NBMS
   PRIM_OUT  = Beam_Vecs )
%NODE( HAMMING
   PRIMITIVE = D_HAMN
   PRIM_IN   = 512,
               1,
               Beam_Vecs THRESHOLD = NBMS*512
   PRIM_OUT  = HAMM_Beams )
%NODE( FFT_512
   PRIMITIVE = D_FFT
   PRIM_IN   = 512,
               NPTSOUT,
               0,
               PT1,
               UNUSED,
               HAMM_Beams THRESHOLD = NBMS*512
   PRIM_OUT  = PSD_cmplx )
%NODE( ENV
%%!!    PRIMITIVE = D_MAG
   PRIMITIVE = D_PWR
   PRIM_IN   = NBMS*NPTSOUT,
               UNUSED,
               PSD_cmplx THRESHOLD = NBMS*NPTSOUT
   PRIM_OUT  = PSD_real, UNUSED)
%NODE( PWR_AVE
   PRIMITIVE = D_AVGN
   PRIM_IN   = NBMS*NPTSOUT,
               NAVE,
               UNUSED,
               UNUSED,
               UNUSED,
               Rep_PSD
                   THRESHOLD = (NAVE*NBMS)*NPTSOUT
                   CONSUME = NBMS*NPTSOUT
   PRIM_OUT  = Beam_Pwr_Acc, UNUSED, UNUSED)
%NODE( PSD_REP1
   PRIMITIVE = D_REPNE
   PRIM_IN   = NBMS*NPTSOUT,
               1,
```

```
                    UNUSED,
                    PSD_real THRESHOLD = NBMS*NPTSOUT
   PRIM_OUT   = FAMILY[Rep_PSD] )
%ENDGRAPH
```

Threat Filter

The Threat Filter processing, shown in Figure 21, consists of replicating the data and then complex demodulating and filtering using three different bandwidths.

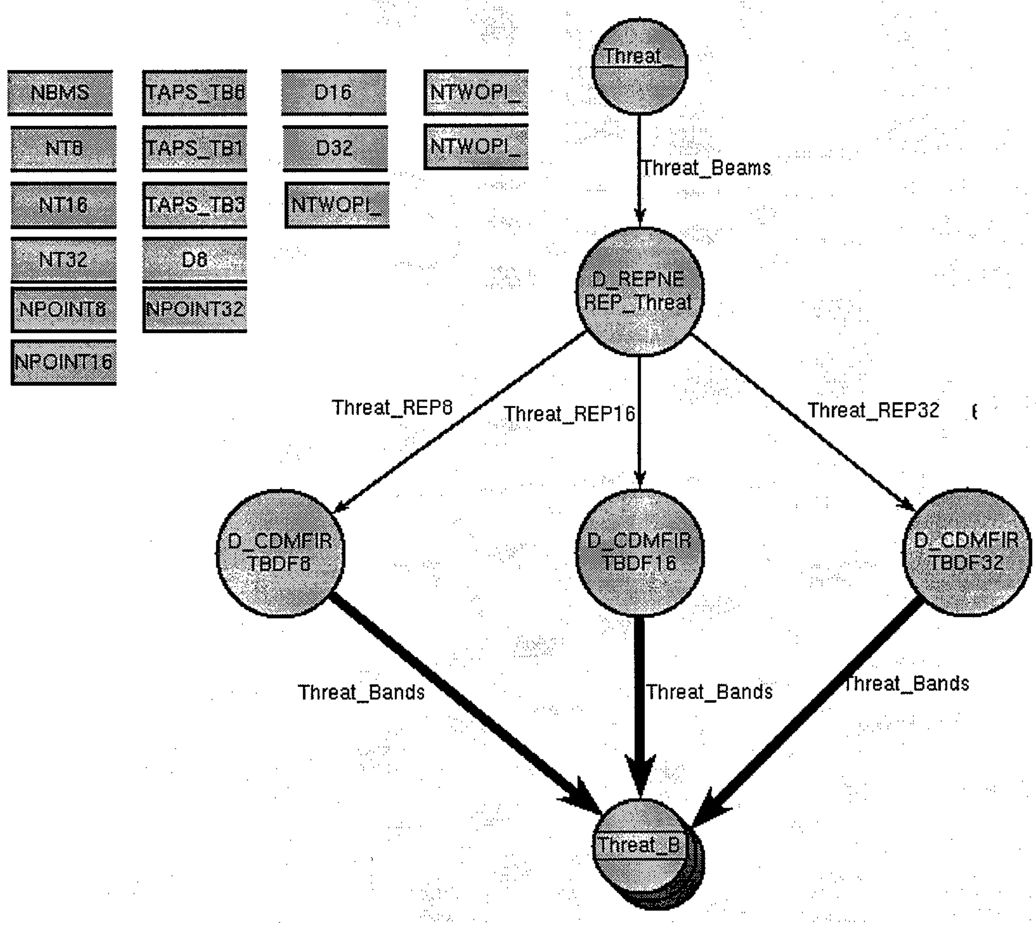

Figure 21. Threat Filter Processing

The SPGN for the Threat Filter processing is:

```
%GRAPH( THREAT_FILT
   GIP      = NBMS : INT
   VAR      = TAPS_TB8 : FLOAT ARRAY(37),
              TAPS_TB16 : FLOAT ARRAY(75),
              TAPS_TB32 : FLOAT ARRAY(151),
              NTWOPI_8 : INT,
              NTWOPI_16 : INT,
              NTWOPI_32 : INT
   INPUTQ  = Threat_Beams : CFLOAT
   OUTPUTQ = [1..3]Threat_Bands : CFLOAT )
%GIP( NT8 : INT INITIALIZE TO 37 )
```

```
%GIP( NT16 : INT INITIALIZE TO 75 )
%GIP( NT32 : INT INITIALIZE TO 151 )
%GIP( D8 : INT INITIALIZE TO 8 )
%GIP( D16 : INT INITIALIZE TO 16 )
%GIP( D32 : INT INITIALIZE TO 32 )
%QUEUE( Threat_REP8 : CFLOAT INITIALIZE TO (NT8-8)*NBMS OF <0.0E0,0.0E0>
)
%QUEUE( Threat_REP16 : CFLOAT INITIALIZE TO (NT16-16)*NBMS OF
   <0.0E0,0.0E0> )
%QUEUE( Threat_REP32 : CFLOAT INITIALIZE TO (NT32-32)*NBMS OF
   <0.0E0,0.0E0> )
%VAR( NPOINT8 : INT INITIALIZE TO 0 )
%VAR( NPOINT16 : INT INITIALIZE TO 0 )
%VAR( NPOINT32 : INT INITIALIZE TO 0 )
%NODE( REP_Threat
   PRIMITIVE = D_REPNE
   PRIM_IN   = 1024*NBMS,
               3,
               UNUSED,
               Threat_Beams THRESHOLD = 1024*NBMS
   PRIM_OUT  = FAMILY[Threat_REP8,Threat_REP16,Threat_REP32] )
%NODE( TBDF8
   PRIMITIVE = D_CDMFIR
   PRIM_IN   = 1024+(NT8-8),
               NBMS,
               0,
               32,
               NTWOPI_8,
               NPOINT8,
               NT8,
               D8,
               TAPS_TB8,
               Threat_REP8
                   THRESHOLD = ((1024+NT8)-8)*NBMS
                   CONSUME = 1024*NBMS
   PRIM_OUT  = [1]Threat_Bands,
               NPOINT8 )
%NODE( TBDF16
   PRIMITIVE = D_CDMFIR
   PRIM_IN   = (1024+NT16)-16,
               NBMS,
               0,
               32,
               NTWOPI_16,
               NPOINT16,
               NT16,
               D16,
               TAPS_TB16,
               Threat_REP16
                   THRESHOLD = ((1024+NT16)-16)*NBMS
                   CONSUME = 1024*NBMS
   PRIM_OUT  = [2]Threat_Bands,
               NPOINT16 )
%NODE( TBDF32
   PRIMITIVE = D_CDMFIR
   PRIM_IN   = (1024+NT32)-32,
               NBMS,
               0,
```

```
                    32,
                    NTWOPI_32,
                    NPOINT32,
                    NT32,
                    D32,
                    TAPS_TB32,
                    Threat_REP32
                        THRESHOLD = ((1024+NT32)-32)*NBMS
                        CONSUME = 1024*NBMS
      PRIM_OUT  =   [3]Threat_Bands,
                    NPOINT32 )
   %ENDGRAPH
```

Threat Spectrum

The Threat Spectrum processing, shown in Figure 22, consists of first corner turning the data to demultiplex the data into a time series for each beam, weighting the data with a Hamming function, Fourier transforming the data, and determining the magnitude of the transformed data. This data is then sent to a display. The data is also averaged over the frequency cells to obtain power averages which are sent to two other processing graphs (which are not part of the demonstration). This processing is performed on each of the three outputs from the different bandwidth Threat Filters.

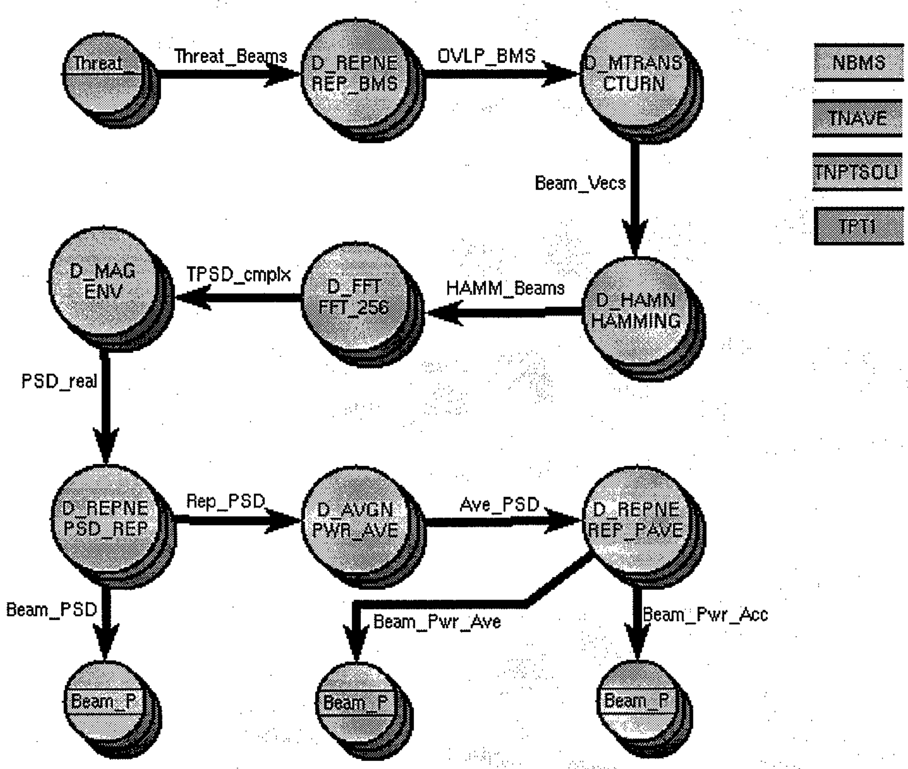

Figure 22. Threat Spectrum Processing

The SPGN for the Threat Spectrum processing is:

```
%GRAPH( Threat_Spectrum
   GIP      = NBMS : INT,
              TNAVE : INT,
              TNPTSOUT : INT,
              TPT1 : INT
   INPUTQ  = [1..3]Threat_Beams : CFLOAT
   OUTPUTQ = [1..3]Beam_Pwr_Acc : FLOAT )
%GIP( DEC_SELECT : INT ARRAY(3) INITIALIZE TO {128, 64, 32} )
%QUEUE( [1..3]Beam_Vecs : CFLOAT )
%QUEUE( [1..3]HAMM_Beams : CFLOAT )
%QUEUE( [1..3]TPSD_cmplx : CFLOAT )
%QUEUE( [1..3]PSD_real : FLOAT )
%QUEUE( [1..3]Rep_PSD : FLOAT INITIALIZE TO (TNAVE-1)*(NBMS*TNPTSOUT) OF
   0.0E0 )
%QUEUE( [1..3]OVLP_BMS : CFLOAT
   INITIALIZE [1]OVLP_BMS TO 128*NBMS OF <0.0E0,0.0E0>
   INITIALIZE [2]OVLP_BMS TO 192*NBMS OF <0.0E0,0.0E0>
   INITIALIZE [3]OVLP_BMS TO 224*NBMS OF <0.0E0,0.0E0> )
%%   II = 1..3
%%   INITIALIZE [II]OVLP_BMS TO (256-(256/(2**II)))*NBMS OF
<0.0E0,0.0E0> )
%%
%NODE( [M=1..3]CTURN
   PRIMITIVE = D_MTRANS
   PRIM_IN   = 256,
               NBMS,
               [M]OVLP_BMS
                   THRESHOLD = 256*NBMS
                   CONSUME = DEC_SELECT(M)*NBMS
   PRIM_OUT  = [M]Beam_Vecs )
%NODE( [M=1..3]HAMMING
   PRIMITIVE = D_HAMN
   PRIM_IN   = 256,
               1,
               [M]Beam_Vecs THRESHOLD = NBMS*256
   PRIM_OUT  = [M]HAMM_Beams )
%NODE( [M=1..3]FFT_256
   PRIMITIVE = D_FFT
   PRIM_IN   = 256,
               TNPTSOUT,
               0,
               TPT1,
               UNUSED,
               [M]HAMM_Beams THRESHOLD = NBMS*256
   PRIM_OUT  = [M]TPSD_cmplx )
%NODE( [M=1..3]ENV
%%!!   PRIMITIVE = D_MAG
   PRIMITIVE = D_PWR
   PRIM_IN   = NBMS*TNPTSOUT,
               UNUSED,
               [M]TPSD_cmplx THRESHOLD = TNPTSOUT*NBMS
   PRIM_OUT  = [M]PSD_real, UNUSED)
%NODE( [M=1..3]PWR_AVE
   PRIMITIVE = D_AVGN
   PRIM_IN   = NBMS*TNPTSOUT,
               TNAVE,
```

```
                 UNUSED,
                 UNUSED,
                 UNUSED,
                 [M]Rep_PSD
                     THRESHOLD = TNAVE*(NBMS*TNPTSOUT)
                     CONSUME = NBMS*TNPTSOUT
   PRIM_OUT   = [M]Beam_Pwr_Acc, UNUSED, UNUSED)
%NODE( [M=1..3]PSD_REP
   PRIMITIVE = D_REPNE
   PRIM_IN   = NBMS*TNPTSOUT,
                 1,
                 UNUSED,
                 [M]PSD_real THRESHOLD = TNPTSOUT*NBMS
   PRIM_OUT   = FAMILY[[M]Rep_PSD] )
%NODE( [M=1..3]REP_BMS
   PRIMITIVE = D_REPNE
   PRIM_IN   = DEC_SELECT(M)*NBMS,
                 1,
                 UNUSED,
                 [M]Threat_Beams THRESHOLD = DEC_SELECT(M)*NBMS
   PRIM_OUT   = FAMILY[[M]OVLP_BMS] )
%ENDGRAPH
```

Simulated Input

The simulated input for the Narrowband Baseline processing is intended to represent the output from a narrowband beamformer. It is assumed that NPT time samples are output from each beam, and that the outputs from all beams are concatenated to form a data set.

The current implementation of the simulated input permits two independent sources or "targets." Each target is represented by a bearing, a signal strength and two tones. For each tone, a relative amplitude and a frequency can be specified.

The generated signal is the summation of the tones from all targets weighted by the tone amplitude, target amplitude, and beam gain for the target based on beam number and the target bearing. The beamformer gain is adjusted for tone frequency. The beam approximation for the top three octaves are the same as in Figure 7. The beams are widened by a factor of two for each lower octave.

If no tones appear in a beam, broadband noise is generated. In the actual implementation, the noise would be bandlimited to some extent by the beamformer.

Typical Output

A typical Frequency-Azimuth display is shown in Figure 23. This display represents the output from Search processing of Octave D. The horizontal axis is frequency. The vertical axis is beam direction with 0 degrees at the bottom and 180 degrees at the top. The simulated signal was two targets, one at 30 degrees and the other at 120 degrees. The target at 30 degrees consists of two

tones, 450 Hz and 600 Hz. The target at 120 degrees consists of two tones, 307 Hz and 600 Hz. These tones can be clearly seen in the display.

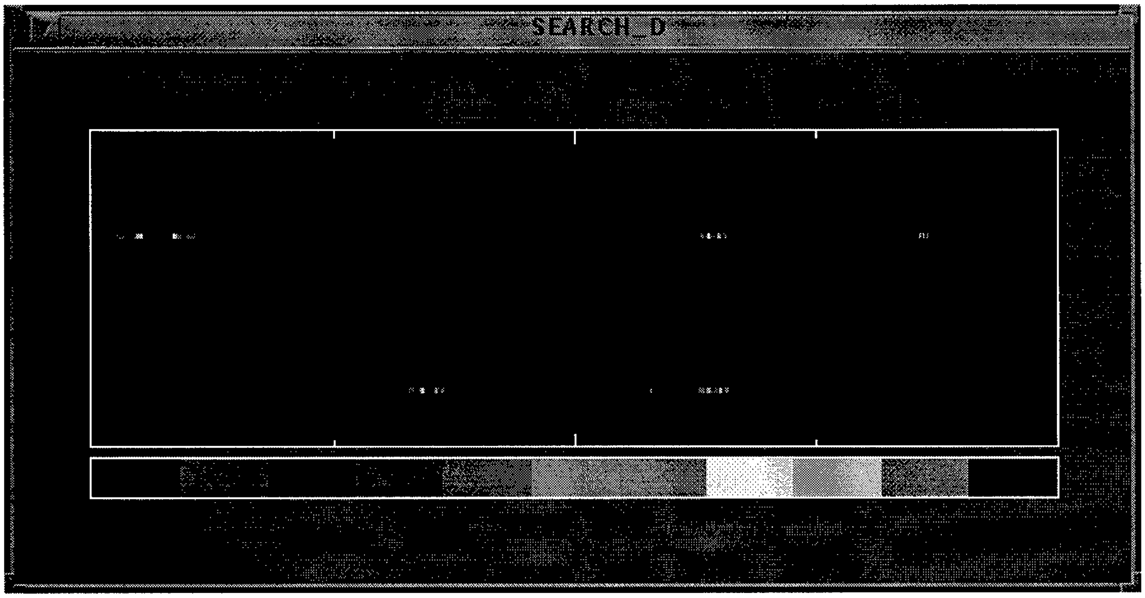

Figure 23. Typical Output for Narrowband Search Processing

Narrowband High Frequency

Overview of the Processing

The Narrowband High Frequency processing is very similar to the Narrowband Baseline Processing limited to five octaves. The top level graph of the processing is shown in Figure 24.

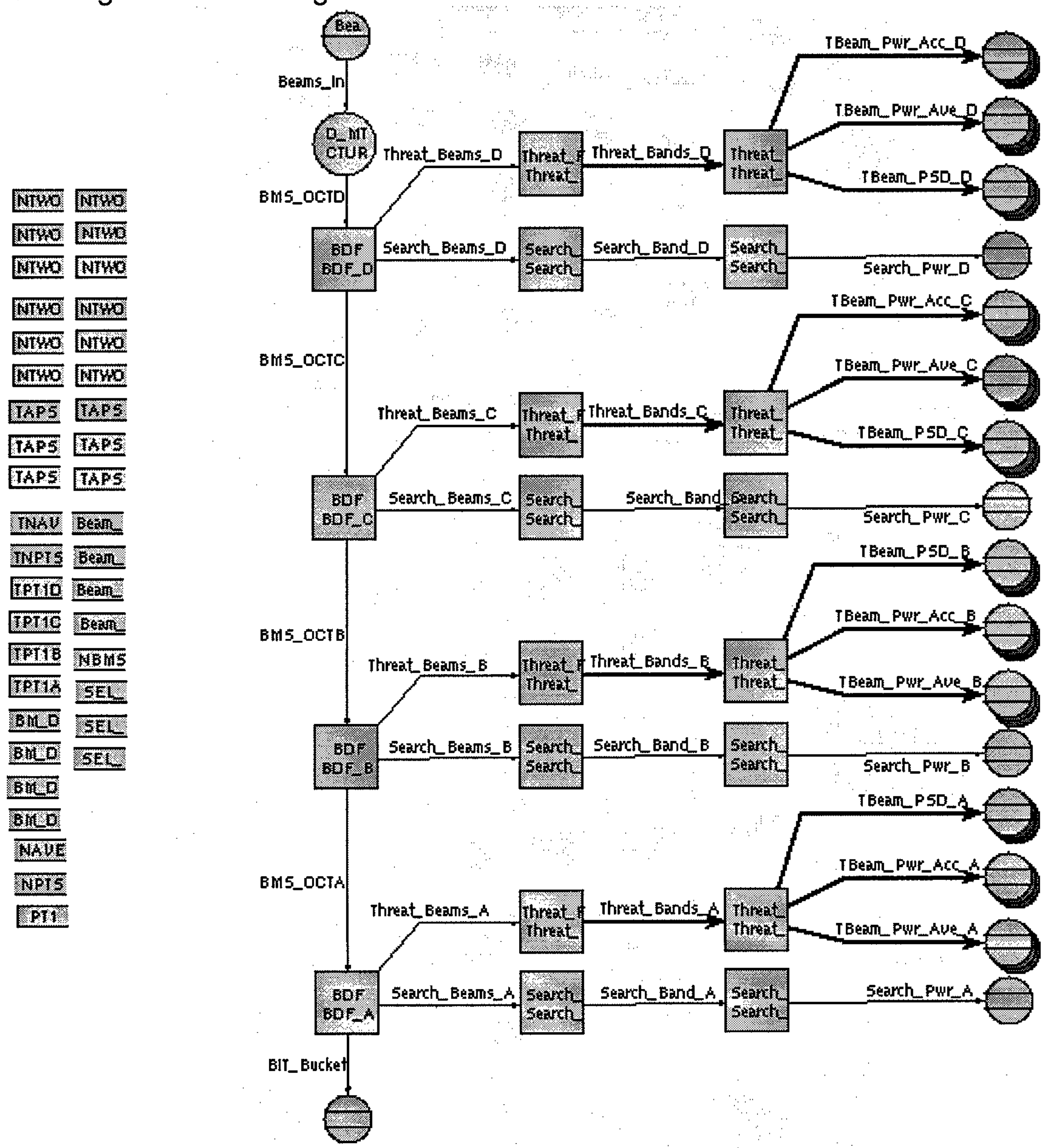

Figure 24. Narrowband High Frequency Processing

Simulated Input

The simulated input for the Narrowband High Frequency processing is the same as the Narrowband Baseline processing.